Optimizing Deep Learning Networks using Multi-Armed Bandits

Salem Abdussalam Ameen

School of Computing, Science and Engineering

College of Science and Technology

University of Salford, Manchester, UK

Submitted in Partial Fulfilment of the Requirements of the Degree of Doctor of Philosophy

December 2017

Contents

1. Introduction 1
1.1. Motivation 1
1.2. Research Problem 3
1.3. Aim and Objectives 3
1.4. Research Methodology 4
1.5. Thesis Organization 8
2. Deep Learning Background 11
2.1. Supervised Learning 11
2.1.1. The Goal of Supervised Learning 12
2.1.2. Regularization 13
2.2. Optimization 13
2.2.1. Gradient Descent 14
2.2.2. Gradient Descent with Momentum 16
2.2.3. Adagrad 17
2.2.4. RMSProp 17
2.2.5. Adam 18
2.3. Back Propagation 18
2.4. Neural Networks 23
2.4.1. Feed Forward Neural Networks 24
2.4.2. Convolutional Neural Networks 25
2.4.3. Recurrent Networks 30
2.4.4. Challenges of Training Neural Networks 33
2.5. Summary 35
3. Literature Review 36
3.1. Background on Pruning Methods 36
3.2. Related Work 39
3.2.1. Pruning Weights 39
3.2.2. Pruning Neurons 42
3.2.3. Pruning Feature Maps 46
3.3. Summary of other Methods for Pruning 48
3.4. Summary 50

4. Multi-Armed Bandit 51
4.1. Notation 51
4.2. Sequential Multi-Armed Bandits 53
4.2.1. Random Explorations 53
4.2.2. Optimistic Explorations 55
4.2.3. Bayesian Bandits 57
4.2.4. Adversarial Bandits 58
4.2.5. Bandits with Multiple Plays 59
4.3. Summary 60
5. Multi-Armed Bandit for Pruning Weights 62
5.1. Architecture of MAB Pruning method 62
5.1.1. Direct Method 66
5.1.2. Epsilon-Greedy Algorithm for Pruning the Weights 68
5.1.3. Win-Stay, Lose-Shift Algorithm for Pruning the Weights 71
5.1.4. UCB1 Algorithm for Pruning the Weights 72
5.1.5. KL-UCB Algorithm for Pruning the Weights 76
5.1.6. Thompson Sampling Algorithm for Pruning the Weights 78
5.1.7. BayesUCB Algorithm for Pruning the Weights 79
5.2. Evaluation 84
5.2.1. Results from the Experiments on the UCI Data sets 85
5.2.2. Results for the MNIST Data set 90
5.3. Summary 92
6. Multi-Armed Bandits for Pruning Neurons 93
6.1. Summary of MAB Algorithm for Pruning Neurons 93
6.1.1. Direct Method 97
6.1.2. Softmax Algorithm for Pruning the Neurons 97
6.1.3. Hedge Algorithm for Pruning the Neurons 98
6.1.4. EXP3 Algorithm for Pruning the Neurons 100
6.2. Evaluation 101
6.2.1. Results from the Experiments on the UCI data sets 104
6.2.2. Testing MAB Based Pruning on Deep Learning Networks 114
6.3. Discussion 125
6.4. Summary 126

7. Multi-Armed Bandits for Pruning Feature Maps 127
7.1. Direct Method 129
7.2. Evaluation 130
7.2.1. Data sets 130
7.2.2. Intial Comparison with the Direct Method 131
7.2.3. Pruning Feature Maps using UCB1 and Thompson Sampling 134
7.3. Discussion 138
7.4. Summary 139
8. Pruning Multiple Neurons and Feature Maps using MABs 140
8.1. The Advantage of Pruning Multiple Neurons over Pruning One 141
8.2. MAB Algorithms for Pruning Multiple Neurons and Featue Maps 141
8.3. Evaluation 144
8.3.1. Testing on UCI Data sets 145
8.3.2. Testing on Deep Learning Data sets 145
8.3.3. Comparing with Pruning Single Neurons or Feature Maps 148
8.4. Summary 149
9. Conclusion and Future Work 150
9.1. Introduction 150
9.2. Summary 150
9.3. Contribution and Main Findings 152
9.4. Future Work 153
10. References 155
11. Appendices 172

List of Tables

TABLE 2-1: EXAMPLE OF A SMALL DATA SET WHERE X_0 AND X_1 ARE FEATURES AND Y IS THE OUTPUT. 19
TABLE 3-1: SUMMARY OF RELATED WORK. 37
TABLE 4-1: A COMPARISON BETWEEN MULTI-ARMED BANDIT ALGORITHMS. 61
TABLE 5-1: CUMULATIVE AVERAGE REWARD FOR BOUNDED REWARDS WHEN PRUNING A WEIGHT. 67
TABLE 5-2: CUMULATIVE AVERAGE REWARD FOR BINARY REWARDS WHEN PRUNING A WEIGHT. 67
TABLE 5-3: WSLS UPDATED THE PROBABILITY P GIVING THE REWARD X AT EACH TRAIL. THE GREEN CELL REPRESENTS THE WEIGHT WITH THE HIGHEST PROBABILITY WHICH WILL BE PLAYED AT THE NEXT. 72
TABLE 5-4: UCB1 METHOD WHERE X IS THE REWARD, N NUMBER OF PLAYS, T IS THE TOTAL PLAYING TIME SO FAR, P_F IS THE PADDING FUNCTION AND W_{JI} IS THE WEIGHTS (THE ALGORITHM WILL CHOOSE THE VALUE). MJI IS CUMULATIVE AVERAGE REWARD AND GREEN COLOUR CELL IS THE ARM WILL BE PLAYED NEXT. 75
TABLE 5-5: RESULTS OF THOMPSON SAMPLING WHERE X IS THE CURRENT BINARY REWARD FOR EACH WEIGHT, T THE TOTAL PLAY TIME, S IS THE SUCCESS, AND F IS FAILURE AND SAMPLE (BETA) IS DRAWN FROM THE BETA DISTRIBUTION FOR EACH WEIGHT. AT EACH TIME STEP, THE ALGORITHM WILL CHOOSE THE WEIGHT THAT HAS THE HIGHEST REWARD AMONG THE OTHERS WHICH IS SHOWN IN THE GREEN CELL. 80
TABLE 5-6: RESULTS OF BAYESUCB ON DIFFERENT PLAY TIME WHERE X IS THE CURRENT BINARY REWARD FOR EACH WEIGHT, T THE TOTAL PLAY TIME, S IS THE SUCCESS, AND F IS FAILURE AND QUANTILE IS DRAWN FROM THE BETA DISTRIBUTION FOR EACH WEIGHT WITH PROBABILITY 1-(1/T). AT EACH TIME STEP, THE ALGORITHM WILL CHOOSE THE WEIGHT THAT HAS THE HIGHEST QUANTILE AMONG THE OTHERS WHICH IS SHOWN IN THE GREEN CELL. 82
TABLE 5-7: UCI DATA SETS 85
TABLE 5-8: COMPUTED ERROR ON VALIDATION DATA SET BEFORE AND AFTER PRUNED THE MODEL. THE GREEN CELL SHOWS THE METHOD WITH LESS ERROR WHILE RED CELL SHOWS THE METHOD WITH LARGE ERROR. THE ARROWS POINT UP IF THE ERROR HIGH, DOWN IF IT IS LOW OR IN RIGHT DIRECTION IF IT IS IN BETWEEN. 87

TABLE 5-9: RESULTS OF THE AVERAGE RANK OF THE METHODS ON 12 DIFFERENT DATA SETS. 88
TABLE 5-10: RUN-TIME PERFORMANCE IN SECONDS FOR THE DIFFERENT PRUNING METHODS ON DIFFERENT DATA SETS. GREEN CELL SHOWS THE METHODS THAT HAVE LESS COMPUTATION TIME WHILE THE RED CELL SHOWS THE ONES WITH THE HIGHEST COMPUTATION TIME. 90
TABLE 5-11: NO OF PARAMETERS IN THE LENET'S MODEL 91
TABLE 5-12: RESULTS OF PRUNING 50% OF TWO LAYERS IN THE LENET'S MODEL. 92
TABLE 6-1: CUMULATIVE AVERAGE REWARD FOR THE BOUNDED REWARDS WHEN PRUNING A NEURON ON EXAMPLE OF DATA AT EACH FORWARD PROPAGATION. 97
TABLE 6-2: SOFTMAX FUNCTION FOR PRUNING THE NEURONS WHERE μi IS CUMULATIVE AVERAGE REWARD, X IS GIVEN REWARD, $\tau = 2$ AND P IS THE PROBABILITY. 98
TABLE 6-3: THE STEPS OF CHOOSING NEXT NEURON TO PRUNE BASED OF HEDGE ALGORITHM. THE GREEN CELL IS THE PROBABILITY OF CHOOSING THE FOLLOWING NEURON. ρ IS THE GENERATED NON-STATIONARY REWARD, W IS THE WEIGHT AND P IS THE PROBABILITY FOR CHOOSING THE NEXT NEURON. $\epsilon = 0.05$. 99
TABLE 6-4: EXP3 FOR PRUNING THE NEURONS WHERE ρ IS THE CURRENT NON-STATIONARY REWARD, W IS THE WEIGHT AND P IS THE PROBABILITY FOR CHOOSING THE NEXT NEURON TO PLAY. Γ=0.1. GREEN CELLS ARE THE NEURONS CHOSEN TO PRUNE. 100
TABLE 6-5: SMALL DATA SET SPECIFICATION. 102
TABLE 6-6: DATA SET SPECIFICATION FOR DEEP LEARNING MODELS. 103
TABLE 6-7: HYPERPARAMETERS OF NEURAL NETWORKS TRAINED ON DIFFERENT DATA SETS. IN ADDITION, THE LEARNING RATE FOR ALL OF THEM IS SET TO 0.001, THE ACTIVATION FUNCTION IS RELU AND THE NUMBER OF EPOCHS IS 100. 103
TABLE 6-8: COMPARISON OF ACCURACY BETWEEN PRUNING BASED ON UCB1 AND DIFFERENT CLASSIFIERS. THE RESULTS WITH RESPECT TO SOME CLASSIFIERS ARE NOT AVAILABLE IN THE CASE OF THE FACE DATA SET BECAUSE OF THE RESOURCE REQUIRED. THE GREEN CELLS INDICATE THAT THE METHOD HAS GOOD ACCURACY IN CONTRAST OF RED CELL. THE ARROWS POINT UP IF THE ERROR HIGH, DOWN IF IT IS LOW OR IN RIGHT DIRECTION IF IT IS IN BETWEEN. 107
TABLE 6-9: RESULTS OF RANKED ACCURACY, F1 SCORE, PRECISION AND RECALL RESULTS BASED ON NEMENYI TEST, WHICH IS USED TO COMPARE THE DIFFERENT MODELS ON 16

DIFFERENT DATA SETS. IN THIS TABLE, THE HIGHEST IS THE BETTER AND THE TABLE IS SORTED ON THE ACCURACY COLUMN. 108
TABLE 6-10: THE RESULT BASED ON THE ACCURACY OF PRUNING DEEP NEURAL NETWORKS ON DIFFERENT DATA SETS USING DIFFERENT ARCHITECTURES. THE TABLE SHOWS THE PRUNED LAYER, NUMBER OF NEURONS IN PRUNED LAYER AND THE PERCENTAGES OF REMOVED NEURONS IN THE LAYER. THE GREEN CELLS INDICATE THAT THE METHOD HAS GOOD ACCURACY IN CONTRAST OF RED CELL. THE ARROWS POINT UP IF THE ERROR HIGH, DOWN IF IT IS LOW OR IN RIGHT DIRECTION IF IT IS IN BETWEEN. 115
TABLE 6-11: RESULTS OF RANKED ACCURACY RESULTS BASED ON NEMENYI TEST, WHICH IS USED TO COMPARE THE DIFFERENT MODELS ON SIX DIFFERENT DATA SETS. 124
TABLE 7-1: EXAMPLES OF COMPUTING FLOPS. 129
TABLE 7-2: RESULT OF PRUNING CONVOLUTIONAL LAYERS. THE GREEN CELLS INDICATE THAT THE METHOD HAS GOOD ACCURACY IN CONTRAST OF RED CELL. THE ARROWS POINT UP IF THE ERROR HIGH, DOWN IF IT IS LOW OR IN RIGHT DIRECTION IF IT IS IN BETWEEN. 136
TABLE 7-3: AVERAGE RANK OF THE ALGORITHMS FOR PRUNING FEATURE MAPS BASED ON ACCURACY, WHERE A HIGHER RANK IS BETTER. 137
TABLE 8-1: PRUNING USING MP-TS AND MP-UCB1. THE GREEN CELLS INDICATE THAT THE METHOD HAS GOOD ACCURACY IN CONTRAST OF RED CELL. THE ARROWS POINT UP IF THE ERROR HIGH, DOWN IF IT IS LOW OR IN RIGHT DIRECTION IF IT IS IN BETWEEN. 147
TABLE 8-2: SUMMARY OF RESULTS BASED ON ACCURACY FROM THREE COMMON DATA SETS THAT WERE USED IN PRUNING BASED ON SINGLE OR MULTIPLE NEURONS OR FEATURE MAPS. CELLS SHADED BLACK INDICATE THERE IS NO RESULT. CELLS SHADED GREEN HAS BEST ACCURACY WHILE RED HAS THE WORSE. 148
TABLE 11-1: RESULTS OF RANKED R SQUARED RESULTS BASED ON NEMENYI TEST, WHICH IS USED TO COMPARE THE DIFFERENT MODELS ON SIX DIFFERENT DATA SETS. 219

List of Figures

FIGURE 1.1: OVERVIEW OF THE THESIS STRUCTURE..... 9
FIGURE 2.1: SGD FLUCTUATION 15
FIGURE 2.2: THE EFFECT OF ADDING MOMENTUM TO GD. 16
FIGURE 2.3: EXAMPLE OF GRAPH OF MULTIPLE NODES. 20
FIGURE 2.4: DEFINITION OF A SINGLE NEURON WITH INPUTS, ACTIVATION FUNCTION AND OUTPUTS. 24
FIGURE 2.5: ARTIFICIAL NEURAL NETWORK WITH ONE HIDDEN LAYER WHERE W IS THE WEIGHT, B IS THE BIAS AND F IS A NON-LINEAR FUNCTION. 25
FIGURE 2.6: CONVNET MODEL WITH TWO INPUTS (INTENSITY AND DEPTH). 26
FIGURE 2.7: LENET MODEL [83]. 29
FIGURE 2.8: ALEXNET MODEL [22]..... 29
FIGURE 2.9: RNN ARCHITECTURE. 30
FIGURE 2.10: LSTM ARCHITECTURE [100]..... 31
FIGURE 2.11: END-TO-END MEMORY NETWORKS. (A): A SINGLE LAYER. (B): A MULTIPLE LAYER..... 32
FIGURE 2.12: NEURAL NETWORKS (A) AFTER DROPOUT (B) AFTER DROPCONNECTION. 34
FIGURE 3.1: A TIMELINE OF RELATED ALGORITHMS. 37
FIGURE 5.1: BLOCK DIAGRAM SHOWS THE MAB TO PRUNE THE WEIGHTS. 64
FIGURE 5.2: THE GENERIC ALGORITHM OF A MAB PRUNING THE WEIGHS. 65
FIGURE 5.3: SYNTHETIC DATA FOR PURPOSE OF EXPLAINING MAB PRUNING ALGORITHMS. 66
FIGURE 5.4: FUNCTION OF EPSILON-GREEDY ALGORITHM TO PRUNE K WEIGHTS. 68
FIGURE 5.5: EPSILON-GREEDY FOR PRUNING 16 WEIGHTS AT DIFFERENT PLAY TIMES. THE RED DOTS DENOTE THE CHOSEN WEIGHT TO PLAYING. THE TOP ONE PLAYED FIRST AND THE BOTTOM ONE PLAYED THE LAST. 70
FIGURE 5.6: FUNCTION OF WSLS BASED ON PURSUIT ALGORITHM TO PRUNE K WEIGHTS.... 71
FIGURE 5.7: FUNCTION OF UCB1 ALGORITHM TO PRUNE K WEIGHTS..... 72
FIGURE 5.8: UCB1 FOR PRUNING 16 WEIGHTS AT DIFFERENT PLAY TIMES. STARTING FROM THE UPPER LEFT TILL THE BOTTOM AT DIFFERENT TIME. THE VERTICAL LINES REPRESENT THE CUMULATIVE AVERAGE REWARD (BOTTOM) AND THE PADDING FUNCTION (TOP). THE RED LINE IS CHOSEN FOR PLAYING. 74
FIGURE 5.9: FUNCTION OF KL-UCB ALGORITHM FOR PRUNING THE K WEIGHTS. 76

FIGURE 5.10: COMPUTE Q OF THE WEIGHT WHERE THE CHARTS ON THE TOP REPRESENT THE WEIGHTS AT THE PLAY TIME BETWEEN (T=49 TO T=64). THEN, THE CHARTS AT THE BOTTOM REPRESENT COMPUTING THE MAXIMUM Q FOR THE CURRENT CHOSEN WEIGHT. 77
FIGURE 5.11: THOMPSON SAMPLING WHERE THERE ARE K WEIGHTS AND wij IS THE WEIGHT SELECTED TO PLAY NEXT. 78
FIGURE 5.12: FUNCTION OF BAYESUCB TO PRUNE K WEIGHTS. 79
FIGURE 5.13: THOMPSON SAMPLING FOR CHOOSING THE ARM TO PLAY NEXT BASED ON THE SAMPLE FROM BETA DISTRIBUTION. THE TWO ARMS ON THE TOP ARE CHOSEN FROM THE FIRST COLUMN IN THE PREVIOUS TABLE WHILE THE CHARTS IN THE BOTTOM ARE CHOSEN FROM THE LAST COLUMN OF THE SAME TABLE. ON THE TOP, THE ALGORITHM WILL CHOOSE THE ARM ON THE LEFT AS IT HAS HIGHER REWARD WHILE ON THE BOTTOM THE ALGORITHM WILL CHOOSE THE ARM ON THE RIGHT AS IT HAS HIGHER REWARD. 81
FIGURE 5.14: BAYESUCB FOR CHOOSING THE ARM TO PLAY NEXT BASED ON THE SAMPLE FROM THE BETA DISTRIBUTION. THE TWO ARMS ON THE TOP ARE CHOSEN FROM THE FIRST COLUMN IN TABLE 5-5 WHILE THE CHARTS AT THE BOTTOM ARE CHOSEN FROM THE LAST COLUMN OF THE SAME TABLE. ON THE TOP, THE ALGORITHM WILL CHOOSE THE ARM ON THE LEFT AS IT HAS HIGHER QUINTILE WHILE ON THE BOTTOM THE ALGORITHM WILL CHOOSE THE ARM ON THE RIGHT AS IT HAS HIGHER QUINTILE..... 83
FIGURE 5.15: COMPARISON OF ALL CLASSIFIERS AGAINST EACH OTHER WITH THE NEMENYI TEST. LINES SHOW THE CRITICAL DIFFERENCE FOR EACH METHOD ANY GROUPS OF CLASSIFIERS THAT ARE NOT SIGNIFICANTLY DIFFERENT (AT P = 0.05) ARE OUT OF THE LINES. THE BLUE DOT SHOWS THE RANK MEAN WHILE THE LINE DETERMINE THE CD WHICH 4.33..... 89
FIGURE 6.1: THE GENERIC ALGORITHM OF A MAB PRUNING NEURONS. 96
FIGURE 6.2: FUNCTION OF SOFTMAX ALGORITHM TO PRUNE K NEURONS. 98
FIGURE 6.3: THE HEDGE FUNCTION FOR PRUNING K NEURONS..... 99
FIGURE 6.4: EXP3 FUNCTION TO PRUNE K NEURONS. 100
FIGURE 6.5: RESULTS OF MAB PRUNING ALGORITHMS. 106
FIGURE 6.6: COMPARISON OF THE ACCURACY OF ALL CLASSIFIERS AGAINST EACH OTHER WITH THE NEMENYI TEST. HORIZONTAL LINES SHOW THE CRITICAL DIFFERENCE AWAY FROM PROPOSED PRUNING METHODS AND ANY OTHER METHODS. GROUPS OF CLASSIFIERS THAT

ARE NOT SIGNIFICANTLY DIFFERENT (P = 0.0.5) ARE OUT OF THE LINES FROM PROPOSED METHODS. CD=8.066. 110
FIGURE 6.7: COMPARISON OF THE F1 SCORE OF ALL CLASSIFIERS AGAINST EACH OTHER WITH THE NEMENYI TEST. HORIZONTAL LINES SHOW THE CRITICAL DIFFERENCE AWAY FROM PROPOSED PRUNING METHODS AND ANY OTHER METHODS. GROUPS OF CLASSIFIERS THAT ARE NOT SIGNIFICANTLY DIFFERENT (P = 0.0.5) ARE OUT OF THE LINES FROM PROPOSED METHODS 111
FIGURE 6.8: COMPARISON OF THE PRECISION OF ALL CLASSIFIERS AGAINST EACH OTHER WITH THE NEMENYI TEST. HORIZONTAL LINES SHOW THE CRITICAL DIFFERENCE AWAY FROM PROPOSED PRUNING METHODS AND ANY OTHER METHODS. GROUPS OF CLASSIFIERS THAT ARE NOT SIGNIFICANTLY DIFFERENT (P = 0.0.5) ARE OUT OF THE LINES FROM PROPOSED METHODS. 112
FIGURE 6.9: COMPARISON OF THE RECALL OF ALL CLASSIFIERS AGAINST EACH OTHER WITH THE NEMENYI TEST. HORIZONTAL LINES SHOW THE CRITICAL DIFFERENCE AWAY FROM PROPOSED PRUNING METHODS AND ANY OTHER METHODS. GROUPS OF CLASSIFIERS THAT ARE NOT SIGNIFICANTLY DIFFERENT (P = 0.0.5) ARE OUT OF THE LINES FROM PROPOSED METHODS. 113
FIGURE 6.10: COMPARISON OF ALL PRUNING ALGORITHMS AGAINST EACH OTHER WITH THE NEMENYI TEST. HORIZONTAL LINES SHOW THE CRITICAL DIFFERENCE AWAY FROM PROPOSED PRUNING ALGORITHMS, THE ORGINAL UNPRUNED MODEL AND TWO OTHER ALGORITHMS THAT ARE NOT SIGNIFICANTLY DIFFERENT (P = 0.0.5) ARE OUT OF THE LINES FROM PROPOSED ALGORITHMS. CD=3.218. 125
FIGURE 7.1 REMOVING THE FILTER $\mathcal{F}_{I,J}$ AND CORRESPONDING FEATURE MAP IN $X_I + 1$ IN CONVNETS. THE TOP DIAGRAM SHOWS THE TWO LAYERS BEFORE PRUNING WHILE THE BOTTOM DIAGRAM SHOWS THE TWO LAYERS AFTER PRUNING THE FILTER AND FEATURE MAP. 128
FIGURE 7.2: CHANGE IN TRAINING LOSS AS A FUNCTION OF THE REMOVAL OF A SINGLE FEATURE MAP FROM THE LENET MODEL. THE FIRST CONVOLUTIONAL LAYER IS ON THE LEFT AND THE SECOND CONVOLUTIONAL LAYER IS ON THE RIGHT. THE TOP ROW SHOWS THE RESULTS FOR BRUTE FORCE PRUNING, THE MIDDLE IS FOR UCB1 PRUNING AND THE BOTTOM IS FOR THOMPSON SAMPLING. 132
FIGURE 7.3: CUMULATIVE REGRET INCURRED ON LENET MODEL TRAINED ON MNIST DATA SET COMPARED TO DIRECT METHOD 134

FIGURE 7.4: COMPARISON OF ALL CLASSIFIERS AGAINST EACH OTHER WITH THE NEMENYI TEST. HORIZONTAL LINES SHOW THE CRITICAL DIFFERENCE AWAY FROM PROPOSED PRUNING ALGORITHMS AND ANY ALGORITHMS. CD=1.133. 138
FIGURE 8.1: PRUNING ALGORITHM BASED ON MP-MAB. 143
FIGURE 8.2: MP-TS FUNCTION WHERE THERE ARE K NEURONS OR FEATURE MAPS. 144
FIGURE 8.3: MP-UCB1 FUNCTION WHERE THERE ARE K NEURONS OR FEATURE MAPS. 144
FIGURE 8.4: PRUNING MULTIPLE NEURONS AT ONE TIME. 146
FIGURE 11.1: VISUALIZATION OF ABALONE DATA. 186
FIGURE 11.2: IMPORTANT FEATURES: ON THE TOP WINE QUALITY DATA SET AND ON THE BOTTOM GLASS DATA SET. 187
FIGURE 11.3: THE CORRELATION BETWEEN VARIABLES FROM LEFT TO RIGHT, PEARSON CORRELATION, SPEARMAN CORRELATION KENDALL CORRELATION. 189
FIGURE 11.4: PCA ON FACE DATA SET. THE 12 PHOTOS ON THE LEFT SHOW SAMPLES FROM THE DATA WHILE THE 12 ON THE RIGHT SHOW IMAGES AFTER APPLYING PCA. 191
FIGURE 11.5: HYPERPARAMETERS OF THE DECISION TREE ON BOSTON HOUSE DATA SET... 192
FIGURE 11.6: FINDING BEST COMBINATION OF PARAMETERS FOR ADABOOST. 194
FIGURE 11.7: ADABOOST ON BOSTON HOUSE DATA SET. 195
FIGURE 11.8: ROC ON ADULT DATA SET. 197
FIGURE 11.9: SOME CONFUSION MATRICES FOR THE PIMA DATA SET 198
FIGURE 11.10: R-SQUARED ON DIFFERENT MAB PRUNED MODELS ON X-AXIS SHOWS THE NUMBER OF PRUNED NEURONS AND IN Y-AXIS SHOWS THE ACCURACY. 215
FIGURE 11.11: R-SQUARED OF MAB ALGORITHMS AND THE ORIGINAL UNPRUNED MODEL TESTED ON REGRESSION TESTING DATA SETS 216
FIGURE 11.12: R-SQUARED BETWEEN MAB PRUNING ALGORITHMS PRUNED 25% OF THE ORIGINAL MODEL AND OTHER REGRESSION MODELS 218
FIGURE 11.13: COMPARISON OF ALL CLASSIFIERS AGAINST EACH OTHER WITH THE NEMENYI TEST. HORIZONTAL LINES SHOW THE CRITICAL DIFFERENCE AWAY FROM PROPOSED PRUNING METHODS AND ANY OTHER METHODS. GROUPS OF REGRESSIONS THAT ARE NOT SIGNIFICANTLY DIFFERENT (P = 0.0.5) ARE OUT OF THE LINES FROM PROPOSED METHODS. CD=10.07 220

Dedication

This PhD Thesis is dedicated to all members of my family:

My parents, who live abroad and kept their hearts with me.

My wife, Asia Ammar, for her support and patience with me throughout my PhD.

My children, Aram, Assal, Aayat and Muhammed as I spent most of my time on research and not as much time with them as I would have liked.

Never give up. Today is hard, tomorrow will be worse, but the day after tomorrow will be sunshine.

Jack Ma

Acknowledgements

First, I would like to express my sincere gratitude to my advisor, Professor Sunil Vadera, for the continuous support in my PhD study and related research, for his patience, motivation, and immense knowledge. His guidance helped me in the research and writing of this thesis and, I could not have imagined having a better advisor and mentor for my study.

Besides my advisor, I would like to thank Professor Tim Ritchings for his time and encouraging words.

Last but not the least, I would like to thank my family, my parents and my brother and sisters for supporting me throughout this project and my life in general.

Declaration

This dissertation is the result of my own work and includes nothing, which is the outcome of work done in collaboration except where specifically indicated in the text. It has not been previously submitted, in part or whole, to any university of institution for any degree, diploma, or other qualification.

Signed: _____

Date: _____

Salem Abdussalam Ameen and full qualifications
Salford

List of Abbreviations and Acronyms

The following table describes the significance of various abbreviations and acronyms used throughout the thesis.

Abbreviation	Meaning
MAB	Multi-Armed Bandit
ConvNets	Convolution Neural Networks
RNN	Recurrent Neural Network
LSTM	Long Short-Term Memory
OBD	Optimal Brain Damage
OBS	Optimal Brain Surgeon
WSLS	Win–Stay, Lose–Shift
UCB	Upper Confidence Bound
KL-UCB	Kullback-Leibler Upper Confidence Bound
BayesUCB	Bayesian Upper Confidence Bound
EXP3	Exponential weight algorithm for Exploration and Exploitation
MP-TS	Multi play Thompson Sampling
MP-UCB1	Multi play UCB1
TS	Thompson Sampling
NN	Neural Networks
E Greedy	Epsilon Greedy
SM	Softmax
G. Prune	Greedy Prune

Abstract

Deep learning has gained significant attention recently following their successful use for applications such as computer vision, speech recognition, and natural language processing. These deep learning models are based on very large neural networks, which can require a significant amount of memory and hence limit the range of applications.

Hence, this study explores methods for pruning deep learning models as a way of reducing their size, and computational time, but without sacrificing their accuracy.

A literature review was carried out, revealing existing approaches for pruning, their strengths, and weaknesses. A key issue emerging from this review is that there is a trade-off between removing a weight or neuron and the potential reduction in accuracy. Thus, this study develops new algorithms for pruning that utilize a framework, known as a multi-armed bandit, which has been successfully applied in applications where there is a need to learn which option to select given the outcome of trials. There are several different multi-arm bandit methods, and these have been used to develop new algorithms including those based on the following types of multi-arm bandits: (i) Epsilon-Greedy (ii) Upper Confidence Bounds (UCB) (iii) Thompson Sampling and (iv) Exponential Weight Algorithm for Exploration and Exploitation (EXP3).

The algorithms were implemented in Python and a comprehensive empirical evaluation of their performance was carried out in comparison to both the original neural network models and existing algorithms for pruning. The existing methods that are compared include: Random Pruning, Greedy Pruning, Optimal Brain Damage (OBD) and Optimal Brain Surgeon (OBS). The thesis also includes an empirical comparison with a number of other learning methods such as KNN, decision trees, SVM, Naïve Bayes, LDA, QDA, logistic regression, Gaussian process classifier, kernel ridge regression, LASSO regression, linear regression, Bayesian Ridge regression, boosting, bagging and random forests. The results on the data sets show that some of the new methods (i) generalize better than the original model and most of the other methods such as KNN and decision trees (ii) outperform OBS and OBD in terms of reduction in size, generalization, and computational time (iii) outperform the greedy algorithm in terms of accuracy.

1. Introduction

1.1. Motivation

Back propagation neural networks have a long history, dating back to the 1980s [1]. These neural networks consist of a network of connected neurons typically organized in layers. Layers are made up of many interconnected neurons which have a nonlinearity function known as an activation. Patterns or examples are presented to the network via the first layer which is known as the input layer, which communicates to one or more middle layer(s) known as the hidden layers where the actual processing is done via a system of weighted connections. The hidden layers then link to the last layer known as an output layer. The decades since their first development has seen many applications [2] of neural networks including in finance [3], a complete check reading system [4], in medical diagnosis [5-8], and in engineering[9-11]. More recently, there has been significant interest in using deep neural networks [12-18]. These deep neural networks consist of a sequence of feature recognition maps, building one layer on top the previous layer and where each layer aims to provide an abstraction of the previous layer, with the final layer performing classification [19]. For example, to recognize objects in images, the first layer aims to learn to recognize edges, the second layer combines edges to form motifs, the third learns to combine motifs into parts, and the final layer learns to recognize objects from the parts identified in the previous layer [20].

Interest in these deep networks grew as a result of their success in the ImageNet Large Scale Visual Recognition Competition (ILSVRC) [1] [21]. In 2012, Krizhevsky et al. [22] demonstrated significant performance improvements over the state of the art in the ImageNet benchmark challenge [21] with their deep network system AlexNet [22]. This has been followed by further advances with deep neural networks such as VGGNet [23], GoogLeNet [24], ResNet [25] and DenseNet [26].

These deep learning networks can become very large; for example, AlexNet has 8 layers and ResNet has 152 layers. Hence this thesis focuses on pruning their size. There are four aspects that motivate the need to prune deep neural networks:

- The first aspect is based on the view that neural networks should aim to mimic the brain to solve problems where each neuron relates to many others. If one accepts this view, which is expressed in most texts on the subject (E.g., Gurney [27]), then it is worth noting how the brain is believed to develop. The number of synapses are very large immediately after the human birth and this number increases sharply after a year from birth. Then, this number is pruned and stabilizes to 500 Trillion at the age of ten [28]. Hence if deep learning is to follow similar steps, they should adopt a pruning step to remove redundant and unimportant weights after developing the large networks [29].
- The second aspect involves using deep neural networks in embedded systems [30]. Currently, most applications of deep learning, such as image detection, natural language processing and speech recognition run on the cloud [31-33]. Running deep neural networks on mobile platforms is difficult at present given the size of the models [31-33].
- The third aspect is that reducing the size of models can speed up the prediction process [34]. This will be especially important for real time applications that use deep learning models [34, 35].
- The fourth aspect to note is that increasing the number of parameters (weights, biases, and neurons) does not necessarily grow the robustness or richness of the learned approximation, but might increase overfitting the data [36].

[1] http://www.image-net.org/challenges/LSVRC/2014/

The primary solution pursued to address the above issues is based on Occam's razor [37]:

> "Assume that an occurrence may have two explanations. Out of these, the simpler would generally be the better one".

The problem addressed in this thesis is how best to do this in a way that reduces the size of the network but does not sacrifice performance.

1.2. Research Problem

Research on neural networks dates to the middle of the 20th century[38] while back propagation neural networks dates to the 1980s [1], and has thrived for many years, so not surprisingly, several techniques have already been developed for pruning neural networks; however, these techniques can be inefficient and very time consuming [39]. In this thesis, the goal is to study and develop algorithms for pruning deep neural networks more efficiently, leading to the following broad questions that need to be addressed:

1. How well do existing algorithms for pruning neural networks perform?
2. Can multi-armed bandit (MAB) best algorithms be developed for pruning and which methods work best?
3. How does the performance of the MAB based pruning methods compare with other methods?

1.3. Aim and Objectives

Having identified the broad questions, the initial phases of research involved surveying the literature on deep learning, understanding existing methods and gaining practical experience with some applications. Practical experience was gained by using various development tools, such as the Torch scientific computing framework [40] to develop a deep network for American Sign Language. This initial work developed a convolutional neural network (ConvNets) aimed at classifying fingerspelling images using both image intensity and depth data. The developed convolutional network was evaluated by applying it to the problem of finger spelling recognition for American Sign Language. This initial work, in itself, produced better results than other published work and led to a journal publication [17]. It also led to a good understanding of deep learning architectures and a key observation that:

pruning deep neural networks involves a trade-off between accuracy and the number of the parameters pruned. That is, as we prune more and more neurons, feature maps or weights, the accuracy may reduce to a point where the network is not useful.
One of the most successful methods for decision making with trade-offs is known as multi-armed bandits [41-56]. Multi-armed bandits provide a framework for studying the exploitation versus exploration dilemma. The scenario for multi-armed bandits involves modelling a gambler who faces a collection of slot machines and needs to select the sequence of machines to be played in order to maximize the rewards. The gambler pulls the arm of a selected machine and receives a reward or not. The goal of the gambler is to maximize the total rewards obtained during a period of playing time. A player needs to choose between an arm that gives the best reward so far (exploitation) or discovering some other arms hoping to find a better arm (exploration).

The aim of this study is to explore if multi-armed bandit algorithms can be used to decide which neurons, feature maps or weights can be removed and lead to efficient neural network models. Given this aim, the research objectives are:

1. To survey and review existing methods for pruning neural networks.
2. To research different multi-armed bandit algorithms that can be adopted for pruning deep neural networks.
3. To utilize multi-armed bandits to develop new methods for pruning deep learning models.
4. To carry out an empirical evaluation of the new multi-armed bandits pruning methods with respect to existing approaches for pruning.

1.4. Research Methodology

Kothari [57] categorises the different types of research based on whether is it descriptive or analytical, applied or fundamental, quantitative or qualitative and conceptual or experimental. These are summarized below based on the exposition in Kothari [57].

Descriptive Research vs. Analytical Research

In descriptive research, the researcher often conducts surveys and enquiries of different kinds for the collection of data. Descriptive research is mainly employed when existing issues need to be addressed or described. This approach finds its application in the fields of social sciences and business and management studies. This method can be differentiated from other methods on the basis that the researcher cannot control the variables; as they are only responsible for reporting events of the past or present. The research projects that undertake this approach are used for the researcher to analyse the existing factors like how frequently a population changes their wardrobe, what brands people prefer, which show has the most viewers etc. All types of survey methods can be classified as descriptive research, including comparative and correlation techniques. However, analytical research is completely different from the former as the researcher has to critically evaluate the material through analysis of given data.

Applied Research vs. Fundamental Research

Research can also be classified as either applied research or fundamental research. In the former, the researcher aims to resolve an immediate problem faced by society or an organization. While, in fundamental or pure research the researcher is dedicated to formulating a theory. Fundamental research is often described as conducting a study with the sole purpose of obtaining knowledge. To give a few examples: research in which human behaviour is studied and related generalizations are made can be classified as fundamental research. Applied research is effective in resolving practical problems at hand; whereas, fundamental research works to formulate theories that will be used as a basis for further studies and have applications at present as well as for the future, and contribute to the body of scientific knowledge.

Quantitative vs. Qualitative

Quantitative research is used for studies that require quantitative analysis to produce the results needed; whereas qualitative research is conducted to establish the existence and/or rationale of a phenomenon. A researcher investigating the reasons for human behaviour, must undertake qualitative research approach. Qualitative research finds its application most commonly in the department of behavioural sciences where studies are done to study the reasons behind human behaviour.

Conceptual vs. Experimental (or Empirical)

Conceptual research is based on abstract ideas or theories. It is most popular among philosophers and thinkers for developing new concepts or for finding new interpretations of those that already exist. In contrast, experimental research is purely based on experiments and/or observations and not much regard is given to a system and theory. Experimental research is a data-based research, where hypotheses are formulated to be verified by observation or experiment. Data must be collected from its source directly in this type of research and standard experimentation for the simulation of desired information must be performed. The researcher is required to have a working hypothesis or guess as to the probable results for initiating this type of research. Their next responsibility is to gather data in favour of or against their hypothesis. Then comes the experimental designs stage, where the materials or subjects are manipulated to obtain the desired information that would prove or disprove the hypothesis. In this type of research, the experimenter has control over the variables being studied and the deliberate manipulation of these variables gives us the results. When a correlation between variables has to be established, empirical research must be used. It is thought that experiments or empirical studies provide the strongest evidence to prove or disprove a given hypothesis.

How to Approach Research?

The above summary suggests that two basic approaches to research exist: namely a quantitative approach and a qualitative approach. In the quantitative approach, the data generated can be analysed to obtain results. There are three sub-categories of the quantitative approach: inferential, experimental and simulation approach. In the inferential approach, a data base is developed, which is then used for determining the features or associations of a population. A sample population is analysed by questioning or merely on the basis of observation for the determination of its characteristics, and these characteristics are then generalized. In the experimental approach, the research environment can be controlled and certain variables can be manipulated to study their relation with other variables, which differentiates it from other types of research. In the simulation approach, an artificial environment is created for fostering relevant information and data to help predict results on the basis of an existing study, which allows the observer to study dynamic behaviour of a system under controlled conditions. The initial conditions, parameters, and exogenous variables, are used to run a simulation study for observing the behaviour of the process over time. Another application of the simulation approach is found in developing models for

predicting results under different conditions. In the qualitative approach the attitudes, opinions and behaviours are analysed subjectively.

In the field of machine learning, where this thesis sits, researchers have mainly utilized experimental and theoretical (fundamental) methods; quantitative approach; and analytical research. Most studies involving algorithm development [58] involve an empirical comparison with respect to other algorithms and utilize an experimental methodology, hence in this study also utilizes this approach. The main steps of this approach are:

1. Carrying out an in-depth literature review on the present methods and techniques to overcome the problem.
2. Design and implement a solution in mind of the problems, which involves devising novel pruning methods that have the ability to prune deep neural networks
3. Empirical evaluation: This involves carrying out many experiments to test the proposed methods.
4. Results analysis: Involves analysing and contrasting the results with similar works in the same domain. A conclusion is established from the findings. The key objective of the developed methods is achieved, with the results seen to outperform the existing works in the same field.

To achieve the desired research goal, namely to develop new algorithms for pruning using MABs that perform well, the following steps are used:

Dataset collection: Most of the modelling approaches in supervised learning fall under the category of data-driven techniques, in which a model learns from human annotation data. It is therefore important to highlight the public available data sets such as the data set from UCI Machine Learning Repository and other different recourses.

Data preparation: For the purposes of this dissertation, the dataset is assumed to be made up of a set of pairs (x, y), where x is an input example and y is a label. The dataset was subsequently divided into three folds, usually a training, validation and test fold (usual percentages could be 60%, 20%, and 20% respectively). However, if the datasets were small then cross validation was used.

Data pre-processing: The convergence of neural networks can be improved by pre-processing the data. For example, standardizing the data (taking off the mean and dividing by the standard deviation individually for every input dimension of x) or subtracting the

mean is amongst the common pre-processing techniques. Besides using the fixed statistics to process the validation and test data, estimating these statistics on the training data is an important activity, as this appropriately simulates the deployment of the final system into a real-world application

Architecture design: For the small data sets, forward neural networks were used to build the model. Image data sets were mostly used for convolutional neural networks and temporal data sets were used with recurrent neural networks.

Optimization: The neural networks were trained and evaluated on a validation dataset. During the training, we monitor the training and validation error. Then the model with the best validation and training error was chosen.

Pruning the model: Once the model was trained then we pruned the model based on the proposed methods and other pruning methods. A preliminary review of the existing work on pruning methods, revealed the following types of methods which were used for comparison:

- Direct methods.
- Regularization and pruning based on magnitude.
- Activation methods.
- First and second order derivative pruning.

Evaluation: The pruned models were evaluated one time on the test set and the accuracy is reported and compared to other pruning techniques. Non-parametric statistical methods are used to validate differences in performance between the various algorithms.

1.5. Thesis Organization

Figure 1.1 presents the structure of the thesis.

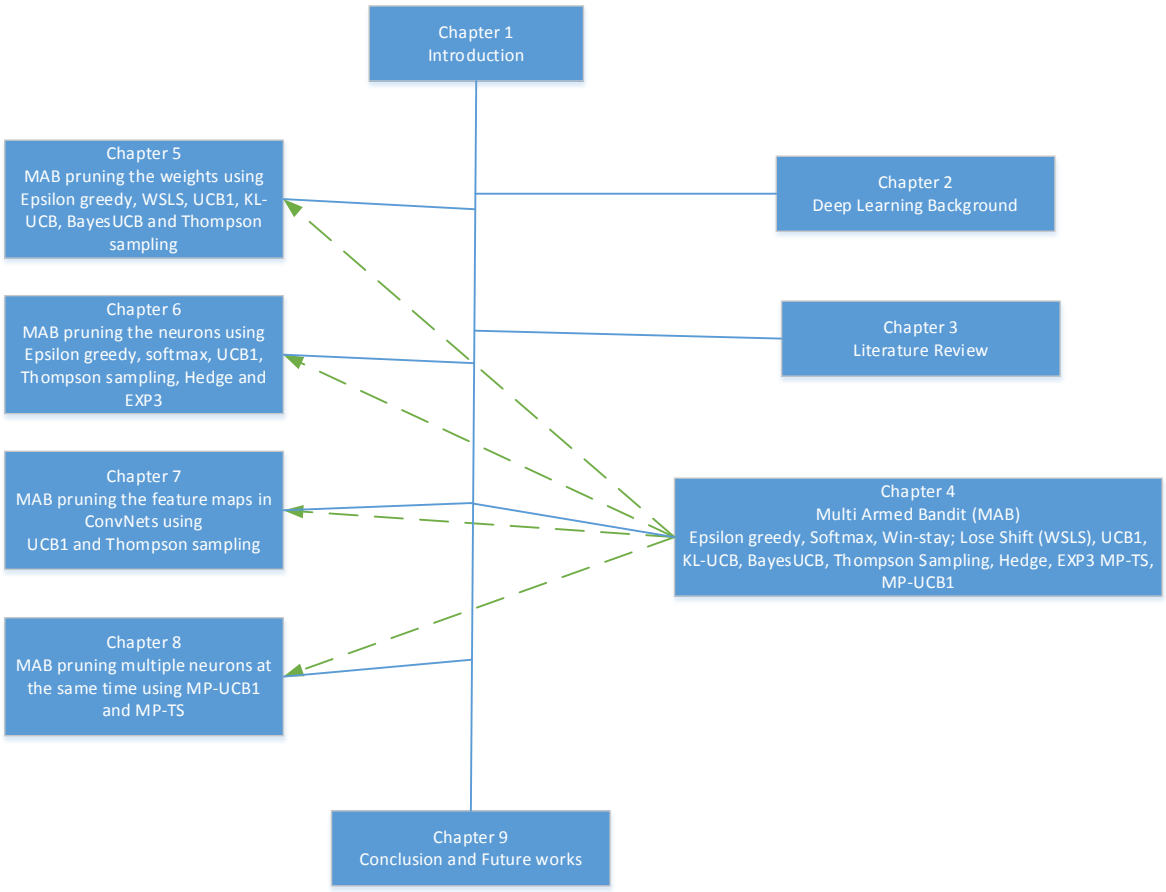


Figure 1.1: Overview of the Thesis Structure.

The following summarizes each chapter of the thesis:

Chapter 2. Deep Learning Background*:* This chapter describes the background and state-of-the-art deep learning models.

Chapter 3. Literature Review: This chapter begins with a brief introduction to the problem of reducing the number of parameters in deep neural networks. After a brief history, details of different kinds of pruning techniques that are identified in the literature survey are reviewed and their strengths and weaknesses presented.

***Chapter 4. Multi-Armed Bandits*:** The literature includes several multi-armed bandit algorithms, each with different characteristics. Chapter 4 introduces the algorithms that are utilized to design and develop the new algorithms for pruning.

Chapter 5. Multi-Armed Bandits for Pruning the Weights: The multi-armed bandits described in Chapter 4 can be utilized either to develop algorithms for pruning neurons, feature maps or weights of a neural network. Chapter 5 describes the use of

MAB methods for pruning the weights. In addition, the chapter presents implementation of MAB pruning algorithms. The implementation is used to evaluate the performance of the MAB pruning algorithms in comparison to each other as well as with existing algorithms.

Chapter 6. Multi-Armed Bandit Algorithms for Pruning the Neurons: This chapter presents the MAB algorithms for pruning neurons and presents the results from comparing the results with state of the art pruning methods.

Chapter 7. Multi-Armed Bandit Algorithms for Pruning Feature Maps: Chapter 7 presents pruning algorithms based on two MAB methods, known as UCB1 and Thompson Sampling, for pruning feature maps and their filter of convolutional layers in ConvNets. The chapter presents the results of pruning the feature maps from ConvNets and comparing the results with the well-known pruning algorithms.

Chapter 8. Multi Armed Bandit Algorithms for Pruning Multiple Neurons and Feature Maps: The previous chapters discussed the use of MAB algorithms for pruning individual weights, neurons or feature maps. This chapter studies the ability of multiple play Thompson Sampling and UCB1 to prune multiple neurons and feature maps at the same time.

Chapter 9. Conclusion and future work: This chapter draws the conclusion and suggests some future work.

2. Deep Learning Background

This chapter presents the background and technical details of different types of neural networks. First, we will begin with a conceptual overview of supervised learning which includes the objective (loss) function and regularization methods. Then, the chapter gives an introduction to optimization and back propagation. Finally, the chapter gives an introduction to feed forward neural networks, convolutional neural networks, and recurrent neural networks. The book by Goodfellow et al. [59] is recommended for a comprehensive and slower-paced overview. In addition, Karpathy [60], Bishop [61] and Abu-Mostafa et al. [62] are the main source for this chapter.

2.1. Supervised Learning

In artificial intelligent, computer programs can be used to map a function *f* between two spaces for example $f: X \rightarrow Y$, where *X* is called an input space and *Y* is known an output space. For instance, in visual recognition, the space of images can be represented as the input *X* and the interval [0, 1] represents the *Y* through which the possibility of an object (like a dog) emerging somewhere in the image is indicated. As another example, in opinion mining, *X* could be the sentence and *Y* could be the opinion of the sentence, such as liked, neutral or disliked. Traditionally, specifying or programming the function *f* explicitly can be difficult for tasks such as image recognition, natural language processing and automatic speech recognition. Supervised learning offers an alternative in which examples $(x, y) \in X \times Y$ of the desired mapping are used to learn the mapping. For our examples, this suggests collecting a data set of images, wherein each may be marked with the absence or presence of a dog, as the same is interpreted by human beings or collecting a data set of sentences from social

media and where each sentence is labelled by carrying either positive or negative meaning [60].

2.1.1. The Goal of Supervised Learning

More formally, a data set of *n* examples is given by $\{(x_1, y_1), \ldots, (x_n, y_n)\}$, where the independent and identically distributed (i.i.d.) samples are utilized to produce these examples from a data generating distribution D; $i.e.(x_i, y_i) \sim D$ for all i [60, 62]. Subsequently, we think about learning the mapping $f: X \rightarrow Y$ by looking for a set of candidate functions, where we attempt to identify the one, which is properly in line with the training examples.

In particular, a class of functions *F* is taken into account. Then, to measure the disagreement between a true label y_i and a predicted label $\hat{y}_i = f(x_i)$ for some $f \in F$, a scalar-valued loss function $L(\hat{y}_i, y)$ is chosen. Finding out $f^* \in F$ that minimizes the expected loss is the goal in learning and formally stated as [62]:

$$f^* = arg \min_{f \in F} E_{(x,y)\sim D} L(f(x), y) \qquad (2.1)$$

Where *argmin* is argument of the minimum which the value of *f* for which the expected loss attains it is minimum. $E_{(x,y)\sim D}$ is the expected loss over the data generating distribution *D*.$\sim$ means the input data is sampled (generated) from the *D*. Since all the possible elements of *D* are not accessible, the optimization in Equation 2.1 is intractable. Hence, the possibility cannot be evaluated or without making idealistically strong assumptions about the form of *L*, *D*, or *f*, we cannot systematically streamline this process. Nonetheless, the expected loss in Equation 2.1 can be estimated with the aid of sampling and can be determined by averaging the loss over the available training data [60]:

$$f^* \approx arg \min_{f \in F} \frac{1}{n} \sum_{i=1}^{n} L(f(x_i), y_i) \qquad (2.2)$$

More specifically, the loss over the available training examples is optimized; however, this is hopefully a good proxy for the actual objective mentioned in Equation 2.1.

2.1.2. Regularization

There can be problems where Equation 2.2 is optimized instead of Equation 2.1. For example, suppose a function f where each x_i in the training data is mapped to its corresponding y_i, however zero is returned everywhere else. We can present this as a solution to Equation 2.2 (for any sensible loss function *L*, where a minimum value is attained when $y = \hat{y}$), but all other points in *D* that are not in the training set would receive a huge loss. More specifically, this function would not be expected to be generalized to all $(x, y) \sim D$. One approach to avoiding this is to introduce a regularization term *R* into the loss function [60]:

$$f^* = \arg\min_{f \in F} \frac{1}{n} \sum_{i=1}^{n} L(f(x_i), y_i) + \frac{\lambda}{n} R(f) \qquad (2.3)$$

Where λ is positive number and there are many types of regularization [63-67], and two that have been widely used are the *L2* and *L1* norms:

$$L2 \text{ norm: } R(f) = \frac{1}{2} \sum_f f^2$$

Where *L2* norm is known as weight decay and is the sum of the squares of all the weights in the network

$$L1 \text{ norm: } R(f) = \sum_f |f|.$$

L1 norm is the sum of the absolute values of the weights:

2.2. Optimization

In the previous section, it was observed that the task of learning a model for a supervised learning problem can be reduced to solving an optimization problem having the form $\theta^* = \arg\min_{\theta} g(\theta)$, where θ is a parameter vector and g normally amalgamates a regularization penalty and the average loss of all examples. The following subsections present the most widely used optimization techniques in neural networks [60].

2.2.1. Gradient Descent

By making additional assumptions about g, the efficiency of the optimization can be enhanced [60]. Specifically, if there is no other option to use except the differentiable functions, a method known as back propagation (its details will be discussed in the next section) would be employed to compute the gradient $\nabla_\theta g$. A vector of partial derivatives is referred to as the gradient, which offers the slope of g along every dimension of θ.

The gradient can be applied as a search direction. We can specifically improve θ (in the sense of attaining lower g), by adding a small amount of the negative gradient. In general, the Gradient Descent (GD) [68, 69] algorithm iterates between the following two steps:

1. The gradient is evaluated.
2. A small step is made in the direction of the negative gradient, the parameters are updated.

In general, the step size λ (also called the learning rate) is a critical parameter in Gradient Descent (GD). The optimization may not converge or even diverge, if the learning rate is too high. Moreover, the learning would become a lengthy process, if it is specified very low [60].

There are three kinds of GD based on how many samples we use to compute the gradient of the loss function [60].

Batch Gradient Descent: Batch GD computes the gradient of the loss function with respect the model's parameters θ. The following steps summarise the batch GD algorithm [68, 69]:

- Estimate the gradient $\nabla_\theta g(\theta)$ with back propagation over all training data set n $\nabla_\theta g(\theta) \approx \nabla_\theta \left[\frac{1}{n}\sum_{i=1}^{n} L(f_\theta(x_i), y_i) + R(f_\theta)\right]$
- Compute the direction $\delta\theta = \lambda\nabla_\theta g(\theta)$ where $\lambda \in R^+$ (positive real number) is learning rate or step size
- Perform a parameter update $\theta_{i+1} = \theta_i - \delta\theta_i$

One problem with batch GD occurs when there is a huge training data set (e.g. there are over 1 million training images in ImageNet). Then, these training data sets cannot fit into the memory during the learning.

Stochastic Gradient Descent (SGD): Instead of computing $\nabla_\theta g(\theta) \approx \nabla_\theta \left[\frac{1}{n}\sum_{i=1}^{n} L(f_\theta(x_i), y_i) + R(f_\theta)\right]$ over all the training data set, SGD [28] computes over one single example in the training data set and is faster than batch GD. One problem with this technique is that SGD performs frequent updates with a high variance that cause the loss function to fluctuate heavily as shown in Figure 2.1.

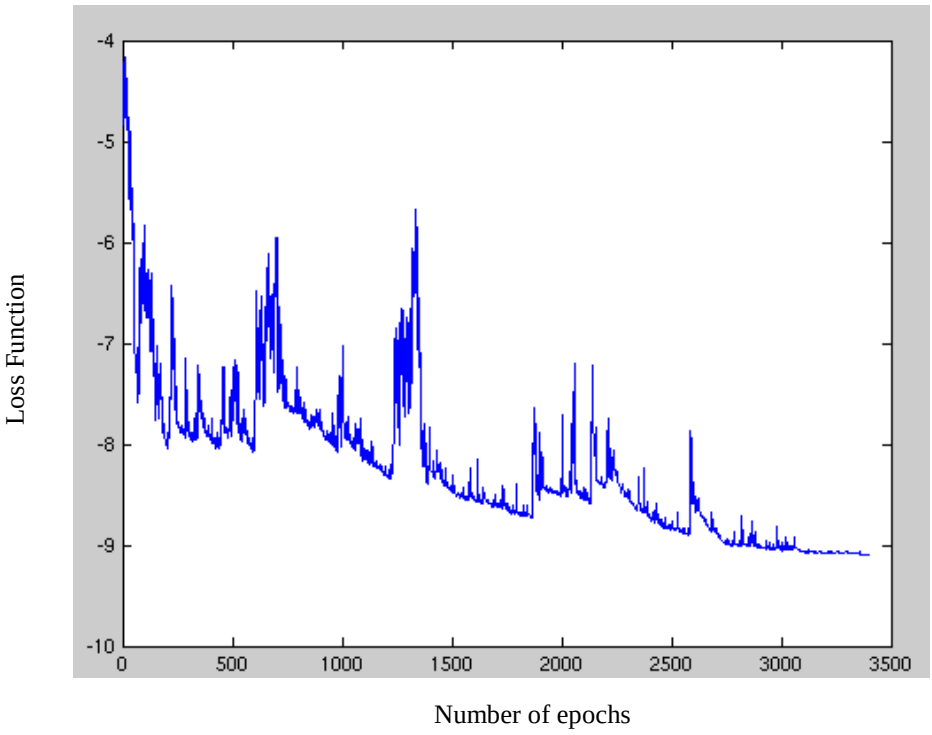


Figure 2.1: SGD fluctuation [2].

Mini-Batch Gradient Descent: In this method, the gradient is estimated through a small mini-batch of examples (e.g. around 200) at a time. As a result, we are enabled to perform a number of approximate updates rather than fewer exact updates. It is an approach, which has excellent functionality / working in most practical applications [70].

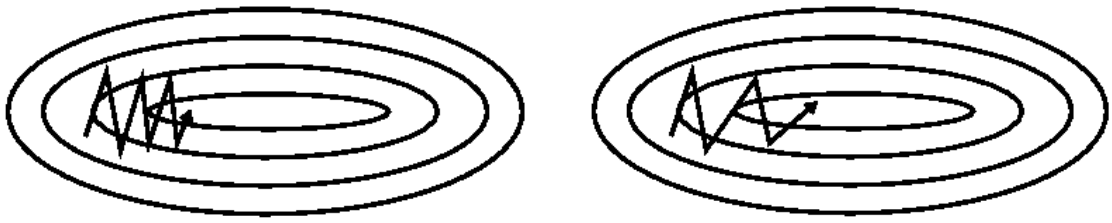
[2] https://upload.wikimedia.org/wikipedia/commons/f/f3/Stogra.png

2.2.2. Gradient Descent with Momentum

The computation of the update direction (Step 2 in GD) can be adjusted and modified to improve the rate of convergence. For example, a method, known as momentum [71], utilizes a proportion of the previous gradients to help maintain a consistent direction thereby often increasing the rate of convergence. The update $\Delta\theta$ is initially computed by updating an intermediate variable $v_{i+1} = \gamma v_i + \lambda \nabla_\theta g(\theta)$ (initialized at zero). It is worth indicating that an exponentially-decaying sum of previous gradient directions is encompassed in the variable *v*. The following steps are the GD with momentum:

- Sample a minibatch of *m* examples from the training data set
- Estimate the gradient $\nabla_\theta g(\theta)$ with back propagation over *m* sampling of training data set *n* $\nabla_\theta g(\theta) \approx \nabla_\theta \left[\frac{1}{m}\sum_{i=1}^{m} L(f_\theta(x_i), y_i) + R(f_\theta)\right]$
- Compute the update direction $\delta\theta = v$ where $v_{i+1} = \gamma v_i + \lambda \nabla_\theta g(\theta)$ and $\gamma \in R$ (real number and practically set to 0.9) and is called momentum.
- Perform a parameter update $\theta_{i+1} = \theta_i - \delta\theta_i$

Figure 2.2 shows the difference between GD with and without momentum. Figure 2.2(a) shows the problem of GD which is that GD oscillates across the slopes of the ravine [72], while Figure 2.2(b) shows how momentum helps accelerate GD in the relevant direction.



(a) GD without momentum (b) GD with momentum

Figure 2.2: The effect of adding momentum to GD[3].

[3] http://ruder.io/optimizing-gradient-descent/index.html#fn:1

2.2.3. Adagrad

Adagrad [73] adapts the learning to the parameters. Adagrad uses the estimate of the first moment of the gradient (the mean). For instance, an intermediate variable *r* is used by Adagrad update [73], where $r_{i+1} = r_i + \nabla_\theta g(\theta) \odot \nabla_\theta g(\theta)$ of sum of squared gradients ($\odot$ is element wise multiplication). Subsequently, the update is modulated by the second moment (the uncentered variance) like this: $\delta\theta = \frac{\lambda}{\delta+\sqrt{r}} \odot \nabla_\theta g(\theta)$, where δ is a small number (e.g. $1e^{-5}$), stopping division by zero [60]. The steps for Adagrad can be summarized as following [36]:

- Sample a minibatch of m examples from the training data set
- Estimate the gradient $\nabla_\theta g(\theta)$ with back propagation over m sampling of training data set *n* $\nabla_\theta g(\theta) \approx \nabla_\theta \left[\frac{1}{m}\sum_{i=1}^{m} L(f_\theta(x_i), y_i) + R(f_\theta)\right]$
- Compute the update direction $\delta\theta = \frac{\lambda}{\delta+\sqrt{r}} \odot \nabla_\theta g(\theta)$ where $r_{i+1} = r_i + \nabla_\theta g(\theta) \odot \nabla_\theta g(\theta)$.
- Perform a parameter update $\theta_{i+1} = \theta_i - \delta\theta_i$

The main advantage of Adagrad is that it does not need to manually update the learning rate throughout the training and in most practical implementations it is to 0.01 [74, 75]. The main disadvantage of Adagrad is growth of the denominator because of accumulation of the squared gradients *r*. This leads the learning rate to shrink over time [76].

2.2.4. RMSProp

A running mean of the second moment is used by the RMSProp update [77] here, $r_{i+1} = \rho r_i + (1-\rho)\nabla_\theta g(\theta) \odot \nabla_\theta g(\theta)$, where ρ is often set to 0.99. The following steps summarize the RMSProp algorithm [36]:

- Sample a minibatch of m examples from the training data set
- Estimate the gradient $\nabla_\theta g(\theta)$ with back propagation over m sampling of training data set *n* $\nabla_\theta g(\theta) \approx \nabla_\theta \left[\frac{1}{m}\sum_{i=1}^{m} L(f_\theta(x_i), y_i) + R(f_\theta)\right]$

- Compute the update direction $\delta\theta = \frac{\lambda}{\delta+\sqrt{r}} \odot \nabla_\theta g(\theta)$ where $r_{i+1} = \rho r_i + (1 - \rho)\nabla_\theta g(\theta) \odot \nabla_\theta g(\theta)$.
- Perform a parameter update $\theta_{i+1} = \theta_i - \delta\theta_i$

2.2.5. Adam

Adam adapts the estimation of both the first and second moments [78] and it can be seen as a mix of RMSProp with momentum. The first moment of the gradients m_g ($m_{g,t}$is m_g *at time t or the current time*) is computed by $m_{g,t+1} = \beta_1 m_{g,t} + (1 - \beta_1)\nabla_\theta g(\theta)$ and the second moment of the gradients v_g ($v_{g,t}$ *is* v_g*at time t*) is computed by $v_{g,t+1} = \beta_2 v_{g,t} + (1 - \beta_2)(\nabla_\theta g(\theta))^2$ where $\beta_1$and β_2 are close to one and practically are set to 0.9 and 0.999 respectively [78]. Subsequently, the update is modulated by these first and second moments thus: $\delta\theta = \frac{\lambda}{\delta+\sqrt{\hat{v}}}\hat{m}$, where δ is a small number (e.g. $1e^{-5}$) where $\hat{m} = \frac{m_g}{1-\beta_1}$ and $\hat{v} = \frac{v_g}{1-\beta_2}$. The following steps summarizes the Adam algorithm [37]:

- Sample a minibatch of *m* examples from the training data set
- Estimate the gradient $\nabla_\theta g(\theta)$ with back propagation over *m* sampling of training data set n $\nabla_\theta g(\theta) \approx \nabla_\theta \left[\frac{1}{m}\sum_{i=1}^{m} L(f_\theta(x_i), y_i) + R(f_\theta)\right]$
- Compute the update direction $\delta\theta = \frac{\lambda}{\delta+\sqrt{\hat{v}}}\hat{m}$, where δ is a small number (e.g. $1e^{-8}$) where $\hat{m} = \frac{m_g}{1-\beta_1}$ and $\hat{v} = \frac{v_g}{1-\beta_2}$
- Perform a parameter update $\theta_{i+1} = \theta_i - \delta\theta_i$

2.3. Back Propagation

The previous section shows that if the gradient of the loss function can be estimated, then GD can be used to reduce it. Back propagation [1, 79-81] is a process where gradients of scalar valued functions are efficiently computed based on their inputs. From calculus, a recursive application of the chain rule is none other than the back propagation algorithm. Remember that g is the main function to calculate the gradients. This function takes the parameters θ and the data set of examples (x_i, y_i) as input.

To understand back propagation, we assume there is an input vector x_0, which is converted through a series of functions $x_i = f_i(x_{i-1})$ where i =1,...,k and the last x_k is a scalar. Then the following steps summarize the back propagation algorithm [60]:

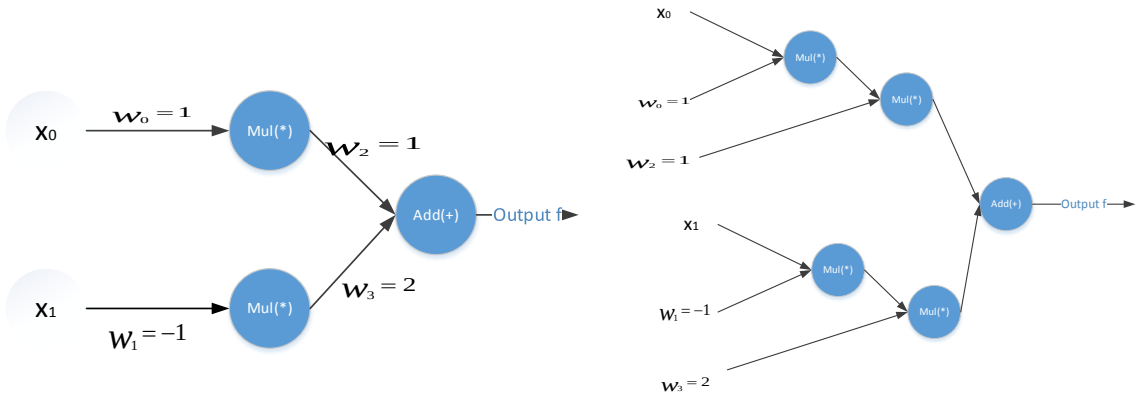
- Compute forward propagation given the input x_0. So $x_1 = f_1(x_0)$, …, $x_i = f_i(x_{i-1})$, …, $x_k = f_k(x_{k-1})$ where f_i are activation functions.
- Using the chain rule, compute the gradient $\frac{\partial x_k}{\partial x_0}$ that will include computing the gradient of all intermediate transformations. These gradients are known as a Jacobian matrix and each transform is given by $\partial x_i / \partial x_{i-1}$.
- Then, the final gradient is given by $\frac{\partial x_k}{\partial x_0} = \prod_{i=1}^{k} \partial x_i / \partial x_{i-1}$. This matrix product of all the Jacobians is used by the GD algorithm as an estimate of the gradient $\nabla_\theta g(\theta)$.

Example: To understand back propagation and GD, consider this example: Assume we have a data set that contains two features x_0 and x_1 and one output y as described in the Table 2-1.

X_0	X_1	y
0	1	1
2	1	4
1	0	2
1	1	1

Table 2-1: Example of a small data set where X_0 and X_1 are features and y is the output.

Consider the graph in Figure 2.3 (a) which consists of three nodes, two of these nodes connect to two external inputs and one node connects to the output. Each connection has an edge as shown in Figure 2.3 where Figure 2.3 (b) represent the operations of the graph where nodes represent the operations.



(a) Graph with 3 nodes, external inputs, and outputs (b) The graph unfolded to explain the operation

Figure 2.3: Example of graph of multiple nodes.

The following steps represent the GD and back propagation in Figure 2.3:

The first step: For simplicity, we will use SGD which applies GD with one example from the training data set that is shown in Table 2-1.

The second step: Compute forward propagation as following:

$$f_1 = x_0 w_0, \frac{\delta f_1}{dx_0} = w_0, \frac{\delta f_1}{\delta w_0} = x_0$$

$$f_2 = x_1 w_1, \frac{\delta f_2}{dx_1} = w_0, \frac{\delta f_2}{\delta w_1} = x_1$$

$$f_3 = f_1 w_2, \frac{\delta f_3}{\delta f_1} = w_2, \frac{\delta f_3}{\delta w_2} = f_1$$

$$f_4 = f_2 w_3, \frac{\delta f_4}{\delta f_2} = w_3, \frac{\delta f_4}{\delta w_3} = f_2$$

$$f = f_3 + f_4, \frac{\delta f}{\delta f_3} = 1, \frac{\delta f}{\delta f_4} = 1$$

Given the third example in the data set and after substituting the values from Table 2-1 and Figure 2.3 we get the following:

$$f_1 = 1 \times 1 = 1$$

$$f_2 = 1 \times -1 = -1$$

$$f_3 = 1 \times 1 = 1$$

$$f_4 = -1 \times 2 = -2$$

$$f = 1 + (-2) = -1$$

Computing the least square error loss function (without regularization for purpose of illustration):

$$L = \frac{1}{2}(y - f)^2 = \frac{1}{2}(1 - (-1))^2 = 2$$

Computing the Jacobian matrix using chain role gives:

$$\frac{\delta L}{\delta f} = -2$$

$$\frac{\delta L}{\delta f_4} = \frac{\delta L}{\delta f}\frac{\delta f}{\delta f_4} = -2 * 1 = -2$$

$$\frac{\delta L}{\delta f_3} = \frac{\delta L}{\delta f}\frac{\delta f}{\delta f_3} = -2 * 1 = -2$$

$$\frac{\delta L}{\delta f_1} = \frac{\delta L}{\delta f}\frac{\delta f}{\delta f_3}\frac{\delta f_3}{\delta f_1} = -2 * 1 = -2$$

$$\frac{\delta L}{\delta w_2} = \frac{\delta L}{\delta f}\frac{\delta f}{\delta f_3}\frac{\delta f_3}{\delta w_2} = -2 * 1 = -2$$

$$\frac{\delta L}{\delta w_0} = \frac{\delta L}{\delta f}\frac{\delta f}{\delta f_3}\frac{\delta f_3}{\delta f_1}\frac{\delta f_1}{\delta w_0} = -2 * 1 * 1 = -2$$

$$\frac{\delta L}{\delta f_2} = \frac{\delta L}{\delta f}\frac{\delta f}{\delta f_4}\frac{\delta f_4}{\delta f_2} = -2 * 2 = -4$$

$$\frac{\delta L}{\delta w_3} = \frac{\delta L}{\delta f}\frac{\delta f}{\delta f_4}\frac{\delta f_4}{\delta w_3} = -2 * (-1) = 2$$

$$\frac{\delta L}{\delta w_1} = \frac{\delta L}{\delta f}\frac{\delta f}{\delta f_4}\frac{\delta f_4}{\delta f_2}\frac{\delta f_2}{\delta w_1} = -2 * 2 * 1 = -4$$

In this example, we assume the parameters of the graph only *W*s then $\nabla_\theta g(\theta) = \nabla_w g(w)$:

$$\nabla_\theta g(\theta) = \nabla_w g(w) = \begin{bmatrix} \frac{\delta L}{\delta w_0} & \frac{\delta L}{\delta w_1} \\ \frac{\delta L}{\delta w_2} & \frac{\delta L}{\delta w_3} \end{bmatrix} = \begin{bmatrix} -2 & -4 \\ -2 & 2 \end{bmatrix}$$

The next step, after computing $\nabla_\theta g(\theta)$ using back propagation involves updating the parameters. However, as described in Section 2.2, there are different GD algorithms and the following presents how each one of them updates the parameters.

<u>Updates using SGD</u>

First, we will start with vanilla SGD. The third step of the algorithm is as following:
$\delta\theta = \lambda\nabla_\theta g(\theta)$ where the learning rate $\lambda = 0.1$ then

$$\delta w_{i+1} = \delta\theta_{i+1} = \lambda\nabla_{w,i}\, g(\theta) = 0.1 * \begin{bmatrix} -2 & -4 \\ -2 & 2 \end{bmatrix} = \begin{bmatrix} -0.2 & -0.4 \\ -0.2 & 0.2 \end{bmatrix}$$

The following is computing the update of the parameters

$$w_{i+1} = \theta_{i+1} = \theta_i - \delta\theta_i = w_i - \delta w_i = \begin{bmatrix} 1 & -1 \\ 1 & 2 \end{bmatrix} - \begin{bmatrix} -0.2 & -0.4 \\ -0.2 & 0.2 \end{bmatrix} = \begin{bmatrix} 1.2 & -0.6 \\ 1.2 & 1.8 \end{bmatrix}$$

GD with momentum at this iteration will give the same result as GD because of $v_{i+1} = \gamma v_i + \lambda \nabla_\theta g(\theta)$ and at the beginning $v = 0$. Then the new $v_{i+1} = \gamma v_i + \lambda \nabla_\theta g(\theta) = 0 * 0.9 + \lambda \nabla_\theta g(\theta) = \lambda \nabla_\theta g(\theta)$ but after this iteration, v will affect the update.

Updates using Adagrad

The following GD is Adagrad, we compute the update direction $\delta\theta = \frac{\lambda}{\delta + \sqrt{r}} \odot \nabla_\theta g(\theta)$ where $r_{i+1} = r_i + \nabla_\theta g(\theta) \odot \nabla_\theta g(\theta)$ and r initializes at 0 then

$$r_{i+1} = r_i + \nabla_\theta g(\theta) \odot \nabla_\theta g(\theta) = 0 + \begin{bmatrix} -2 & -4 \\ -2 & 2 \end{bmatrix} \odot \begin{bmatrix} -2 & -4 \\ -2 & 2 \end{bmatrix} = \begin{bmatrix} 4 & 16 \\ 4 & 4 \end{bmatrix}$$

Then

$$\delta\theta = \frac{\lambda}{\delta + \sqrt{r}} \odot \nabla_\theta g(\theta) = \delta\theta = \frac{0.1}{0.00005 + \sqrt{\begin{bmatrix} 4 & 16 \\ 4 & 4 \end{bmatrix}}} \odot \begin{bmatrix} -2 & -4 \\ -2 & 2 \end{bmatrix} = \begin{bmatrix} -0.025 & -0.006 \\ -0.025 & 0.025 \end{bmatrix}$$

Then, perform a parameter update

$$\theta_{i+1} = \theta_i - \frac{\lambda}{\delta + \sqrt{r}} \odot \nabla_{\theta,i} g(\theta) = \begin{bmatrix} 1 & -1 \\ 1 & 2 \end{bmatrix} - \begin{bmatrix} -0.025 & -0.006 \\ -0.025 & 0.025 \end{bmatrix} = \begin{bmatrix} 1.025 & -0.99 \\ 1.025 & 1.98 \end{bmatrix}$$

Updates using RMSProp

The next GD is RMSProp, we compute the update direction $\delta\theta = \frac{\lambda}{\delta + \sqrt{r}} \odot \nabla_\theta g(\theta)$ where $r_{i+1} = \rho r_i + (1 - \rho) \nabla_\theta g(\theta) \odot \nabla_\theta g(\theta)$ and r_i initializes at 0 then

$r_{i+1} = \rho r_i + (1 - \rho) \nabla_\theta g(\theta) \odot \nabla_\theta g(\theta) = 0.99 * 0 + (1 - 0.99) *$

$\begin{bmatrix} -2 & -4 \\ -2 & 2 \end{bmatrix} \odot \begin{bmatrix} -2 & -4 \\ -2 & 2 \end{bmatrix} = \begin{bmatrix} 0.04 & 0.16 \\ 0.04 & 0.04 \end{bmatrix}$ Then

$$\delta\theta = \frac{\lambda}{\delta + \sqrt{r}} \odot \nabla_\theta g(\theta) = \frac{0.1}{0.00005 + \sqrt{\begin{bmatrix} 0.04 & 0.16 \\ 0.04 & 0.04 \end{bmatrix}}} \odot \begin{bmatrix} -2 & -4 \\ -2 & 2 \end{bmatrix} = \begin{bmatrix} -0.249 & -0.063 \\ -0.249 & 0.250 \end{bmatrix}$$

Then, perform a parameter update

$$\theta_{i+1} = \theta_i - \frac{\lambda}{\delta + \sqrt{r}} \odot \nabla_\theta g(\theta) = \begin{bmatrix} 1 & -1 \\ 1 & 2 \end{bmatrix} - \begin{bmatrix} -0.249 & -0.063 \\ -0.249 & 0.250 \end{bmatrix} = \begin{bmatrix} 1.25 & -0.94 \\ 1.25 & 1.75 \end{bmatrix}$$

Updates using Adam

The last GD is Adam, we compute the update direction $\delta\theta = \frac{\lambda}{\delta+\sqrt{\hat{v}}}\hat{m}$, where δ is a small number where $\hat{m} = \frac{m_g}{1-\beta_1}$ and $\hat{v} = \frac{v_g}{1-\beta_2}$, $m_{g,i+1} = \beta_1 m_{g,i} + (1-\beta_1)\nabla_\theta g(\theta)$ and $v_{g,i+1} = \beta_2 v_{g,i} + (1-\beta_2)(\nabla_\theta g(\theta))^2$

$$m_{g,i+1} = \beta_1 m_{g,i} + (1-\beta_1)\nabla_\theta g(\theta) = 0.9 * 0 + (1-0.9)\begin{bmatrix}-2 & -4\\ -2 & 2\end{bmatrix} = \begin{bmatrix}-0.2 & -0.4\\ -0.2 & 0.2\end{bmatrix}$$

$$v_{g,i+1} = \beta_2 v_{g,i} + (1-\beta_2)(\nabla_\theta g(\theta))^2 = 0.999 * 0 + (1-0.999)(\begin{bmatrix}-2 & -4\\ -2 & 2\end{bmatrix})^2$$

$$= \begin{bmatrix}0.004 & 0.016\\ 0.004 & 0.004\end{bmatrix}$$

$$\hat{m} = \frac{\begin{bmatrix}-0.2 & -0.4\\ -0.2 & 0.2\end{bmatrix}}{1-0.9} = \begin{bmatrix}-2 & -4\\ -2 & 2\end{bmatrix}$$

$$\hat{v} = \frac{\begin{bmatrix}0.004 & 0.016\\ 0.004 & 0.004\end{bmatrix}}{1-0.999} = \begin{bmatrix}4 & 0.16\\ 4 & 0.0044\end{bmatrix}$$

Then, computing the update direction $\delta\theta = \frac{\lambda}{\delta+\sqrt{\hat{v}}}\hat{m}$

$$\delta\theta = \frac{\lambda}{\delta+\sqrt{\hat{v}}}\hat{m} = \frac{0.1}{0.00001 + \sqrt{\begin{bmatrix}4 & 0.16\\ 4 & 0.0044\end{bmatrix}}}\begin{bmatrix}-2 & -4\\ -2 & 2\end{bmatrix} = \begin{bmatrix}-0.025 & -0.006\\ -0.025 & 0.025\end{bmatrix}$$

Finally, perform a parameter update

$$\theta_{i+1} = \theta_i - \frac{\lambda}{\delta+\sqrt{r}}\odot\nabla_\theta g(\theta) = \begin{bmatrix}1 & -1\\ 1 & 2\end{bmatrix} - \begin{bmatrix}-0.025 & -0.006\\ -0.025 & 0.025\end{bmatrix} = \begin{bmatrix}1.025 & -0.994\\ 1.025 & 1.975\end{bmatrix}$$

Finally, from the previous example, we can notice that SGD has less hyperparameters while Adam has the most.

2.4. Neural Networks

In the previous sections, it was observed that arbitrary differentiable functions *f* can be defined, which map the inputs *x* to predicted outputs $\hat{y}$, and that a GD procedure can be used to optimize a differentiable loss function. The function *f*, which has been left unspecified until now, will now be discussed.

Figure 2.4 shows a neural network with one neuron. In this example, the neuron has three inputs (x_1, x_2 and x_3) and one output. Each neuron is connected by weights to the other

neurons and these connections are called weights. The weight $w_{ji}^{(l)}$ is the connection at layer l and is connected between neuron j in the input and neuron i in the layer l. Each neuron has an activation function which is denoted by $f(x)$ notation.

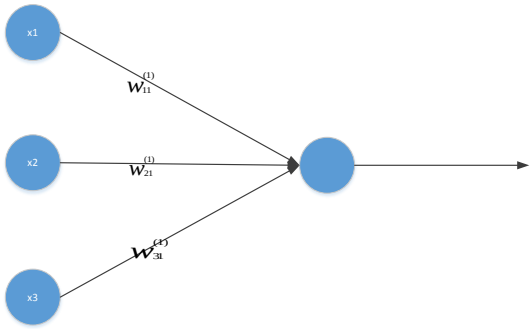


Figure 2.4: Definition of a single neuron with inputs, activation function and outputs.

Such single neurons provide the building blocks for different architectures for neural networks. There are several different types of neural networks, namely: feed forward networks, which have been widely used for classification [82]; Convolutional networks (ConvNets) [83] which have been utilized mainly for image processing tasks [22-25, 83-87] and Recurrent networks that take account of previous states [88]. The following subsections describe these in more detail. There are, of course many other types such as unsupervised learning networks such as Kohonen networks [89], the Boltzmann machine networks [90], deep belief networks [91], generative adversarial networks [92] and autoencoder networks [93, 94] which are not the focus of this thesis. Readers interested in other types of networks are therefore referred to [59].

2.4.1. Feed Forward Neural Networks

A feed forward network consists of layers of neurons, with each layer is connected to the following layer of neurons. Figure 2.5 presents an example of a feed forward network which has an input layer with three neurons *x1*, *x2*, *x3*, a hidden layer with three neurons and a single output.

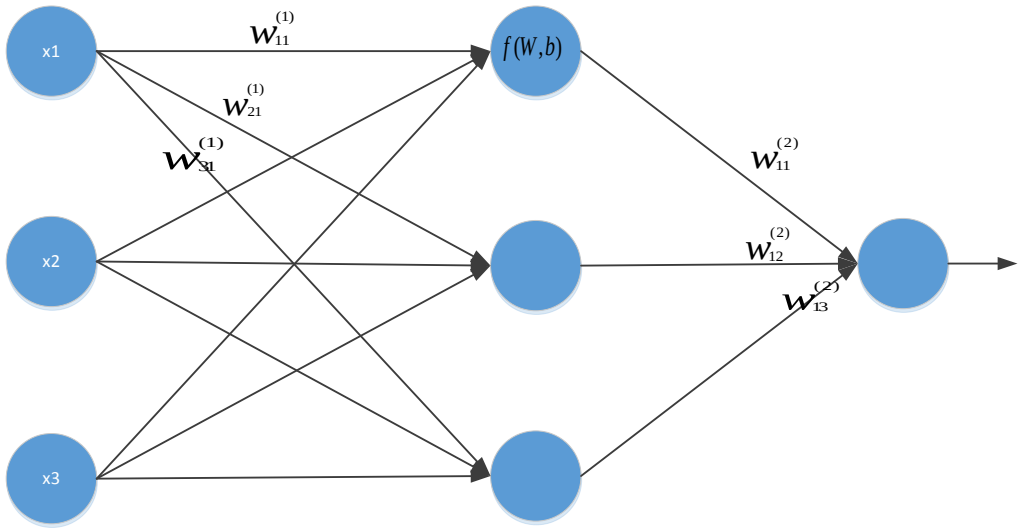


Figure 2.5: Artificial neural network with one hidden layer where *W* is the weight, *b* is the bias and *f* is a non-linear function.

In this network, W_1 is a matrix of the weights between the input and hidden layer, and W_2 is a matrix of weights between the hidden layer and the output neuron.
A 2-layer neural network like the one in this figure will have an output: $f(x) = W_2 f(W_1 x)$, Where f is an element-wise non-linearity (e.g. sigmoid) and *W1*,*W2* are matrices. A three-layered network will have an output: $f(x) = W_3 f(W_2 f(W_1 x))$.

The entire neural network is shortened (in representational power) to a linear function, if the non-linearity is an identity function. The sigmoid function $f(x) = \frac{1}{1+e^{-x}}$, tanh function $f(x) = tanh(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$ and the rectified linear neuron function (ReLU) $f(x) = max(0, x)$ are thought to be the common settings for the non-linearity. In addition, Leaky ReLU function [95] adds a small negative slope to ReLU function as follows: $f(x) = \begin{cases} x & if\ x \geq 0 \\ \alpha x & otherwise \end{cases}$, where α is small constant and set to 0.01. Maxout networks function [96] is another generalization of ReLU as given by $\max(w_1^T x + b_1, w_2^T x + b_2)$ where it doubles the number of weights in every neuron. It is worth indicating that the non-linearity is not usually included in the last layer of the neural network. Moreover, a simple linear transformation is none other than a 1-layer neural network.

2.4.2. Convolutional Neural Networks

Utilizing fully connected networks of the type illustrated in Figure 2.5, for applications such as face recognition and speech understanding, is computationally intractable given the large number of parameters. Convolutional Neural Networks (CNNs, or ConvNets) [83] were developed specifically for these types of applications, where the data has some spatial topology (e.g. videos, images, character sequences in text, sound spectrograms in speech processing or 3D voxel data). A ConvNet typically takes a multi-dimensional array (i.e. a tensor) as input and produces a classification as an output [60]. For instance:

- In an image processing application, the input could be images represented by a 32×32×3 tensor[4] (for 3 colour channels red, green, blue) or represented by 32×32×1 tensor (for 1 grey channel).
- In a speech recognition system, an array of size 1000 ×128 could be used for a sound spectrogram, which depicts the amplitude of any one of 128 frequencies at any interval/stage from *t = 1* to *t=1000*.

Figure 2-1 gives an example of a ConvNets from Ameen & Vadera [17] which is described below.

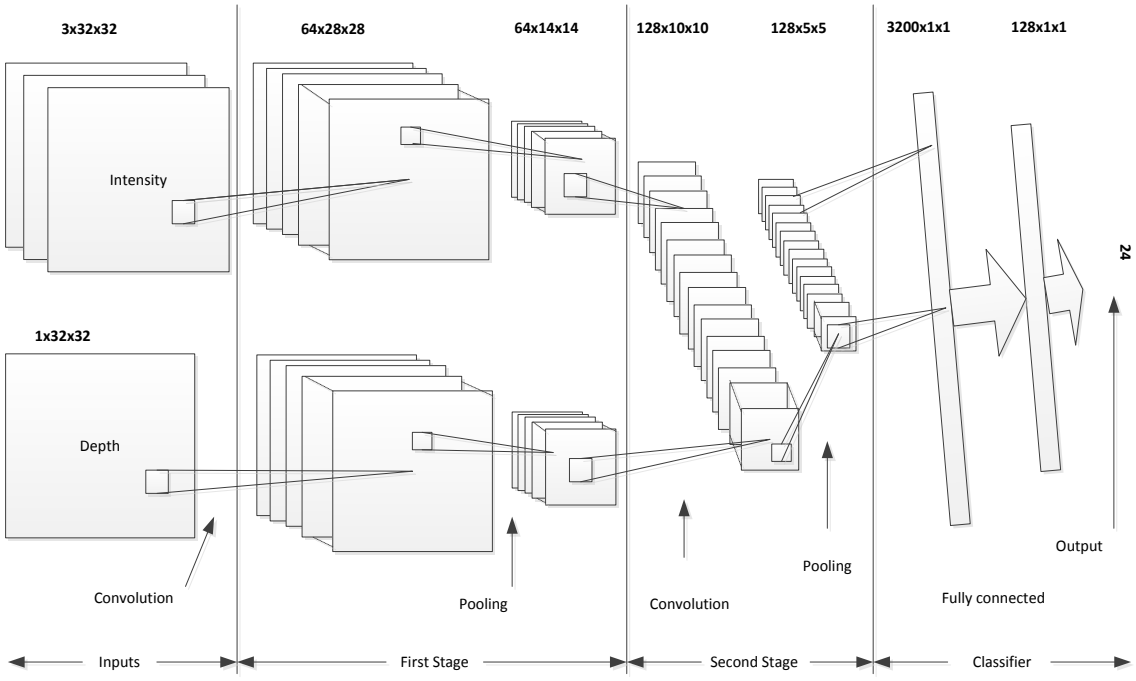


Figure 2.6: ConvNet model with two inputs (Intensity and Depth).

[4] Tensor is a mathematical object that can be used to describe physical properties, just like scalars and vectors

Convolutional Layer: The Convolutional Layer (or the CONV layer) is the core computational building block of a Convolutional Neural Network, where an input tensor is taken to deliver an output tensor. This is done through convolving the input with a set of filters. In general, a convolutional layer is considered for images (i.e. assuming input tensors with three spatial dimensions) as following [60]:

(a) The input is a tensor of size $W_1 \times H_1 \times C_1$.

(b) It needs four hyper-parameters, which are: the number of filters[5] C_k, their spatial extent *K*, the amount of zero padding on the borders of the input *P*, and the stride with which they are applied *S*.

(c) An output volume of size $W_2 \times H_2 \times C_2$ is generated by the convolutional layer, where $W_2 = \frac{W_1 - F + 2p}{s} + 1$, $H_2 = \frac{H_1 - F + 2p}{s} + 1$, and $C_2 = C_k$ (number of filters equal number of output feature maps).

(d) Total number of weights is $(F * F * C_1 * C_k)$ and total number of biases is C_k while $(F * F * C_1)$ represents the number of weights in each filter where there are C_k filters.

Given below is the analysis of convolutional layer. The (pre-activation) output of one neuron, which has connections to that particular chunk of the input array is represented by the result of a dot product with one filter at one specific location. Moreover, a parameter sharing scheme is introduced, where all the same weights are used by the neighbouring neurons in one activation map, since each filter is slid over the input besides using the same weights at every location. In each convolutional layer, there is a considerable decrease in the number of parameters, through which overfitting is addressed.

For the example in Figure 2.6, $32 \times 32 \times 3$ and $32 \times 32 \times 1$ inputs are processed with a convolutional layer with 64 filters having size $5 \times 5 \times 3$ and $5 \times 5 \times 1$ respectively (parameters that we want to learn), and they employ a stride of 1 and padding of 0. In this case, the output would be $28 \times 28 \times 64$ in both sides, which represents the firing of all filters at all spatial locations. To appreciate the difference between a convolution layer and a fully connected layer, it is worth computing the number of parameters required for Figure 2-6. For the convolution layer, there are 5*5*3*64 weights and 64 biases in top first convolution layer and 5*5*1*64 weights and 64 biases in the bottom first convolution layer. Both, the top and

[5] A filter and kernel are similar concepts, we therefore use them interchangeably throughout this thesis.

the bottom convolution layer, produce 25,088 outputs. On the other hand, if the same number of neurons in the hidden layer joins this fully connected layer, we would be using $25088 * (28 * 28 * 3 + 1) = 59{,}032{,}064$ and $25088 * (28 * 28 * 1 + 1) = 1{,}969{,}408$ parameters (biases and weights) - a very huge number, and it is anticipated to deeply overfit even if we assumed that the results could be computed or stored.

Pooling layers: Besides convolutional layers, use of pooling layers to control overfitting is a common practice. Pooling reduces the size of the representation with a fixed down sampling transformation Specifically, each channel (activation map or feature map)[6] independently operates the pooling layers and they are then down sampled in a spatial manner. For example, 2×2 filters with a stride of 2 can be applied, where the max operation (i.e. over 4 numbers) is computed by each filter. As an outcome, a factor of 2 is used to downscale an input tensor in both width and height. In addition, a factor of 4 is used to reduce the representation size by losing some spatial data [60]. In Figure 2.6, both convolutional layers are pooled from 28×28×64 to 14×14×64 by using max pooling with stride 2.

ConvNet architectures: To conclude, the convolutional layers are stacked to build a convolutional network. Moreover, pooling layers are used to reduce the computational complexity of the architecture [60]. For the example in Figure 2.6, 64 filters (feature maps) are used, each with a 5x5 receptive field[7], no zero padding and a stride of one which leads to 64 planes each of dimension 28x28. In the second stage, 128 filters with the same receptive field and stride are used, leading to an array of 128x10x10. Each single number in this dimension is squashed using a *Tanh* as an activation function. In the first stage, a pooling operation is applied to reduce the impact of translations and reduce the number of weights that would be needed. In this example, the 64 filters are pooled by a 2x2 receptive field with a stride of 2, leading to 64 planes each of dimension 14x14. In the second stage, the 128 filters are pooled by a 2x2 receptive field with stride of 2, leading to 128x5x5 planes.

LeNet model: This is considered to be the first ConvNet [83]. The first layer is the input layer with width $W_1 = 28$ to make a size of 28x28. This is followed by a convolution layer,

[6] A channel and feature map are similar concepts, we therefore use them interchangeably throughout this thesis.

[7] The receptive field and filter size are similar concepts, we therefore use them interchangeably throughout this thesis

which is a 5x5 receptive field (kernel with F=5) with no zero padding (P=0) and stride (S=1) to give $W_2 = \frac{W_1 - F + 2*P}{S} + 1 = \frac{28-5+2*0}{1} + 1 = 23 + 1 = 24$ with 24x24 output and four feature maps. It utilises 2x2 average pooling and a stride of 2 to reduce the size to $\frac{24-2}{2} + 1 = 12$.

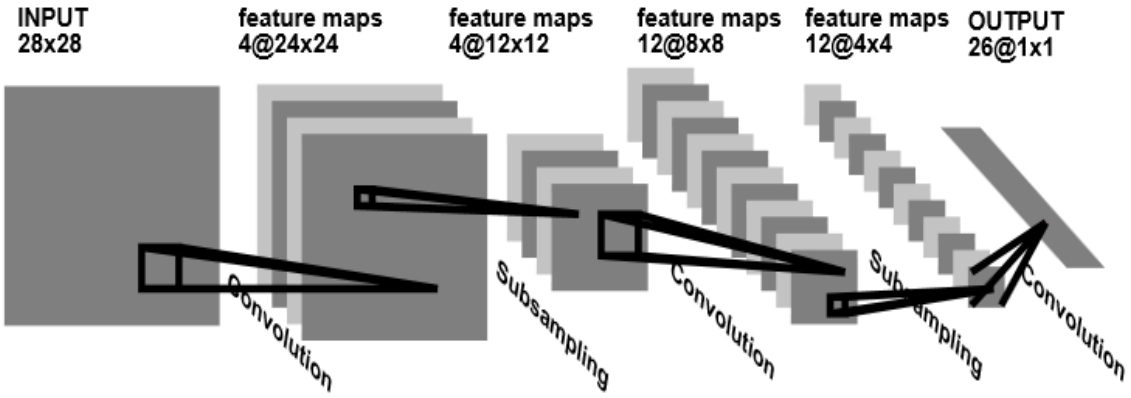


Figure 2.7: LeNet Model [83].

AlexNet model: AlexNet [22] is the first successful framework in the ImageNet Large Scale Visual Recognition Competition (ILSVRC)[8] challenge. This is a modified version of the LeNet model, which allows concatenation of two convolution layers followed with nonlinearity without the need for pooling layers.

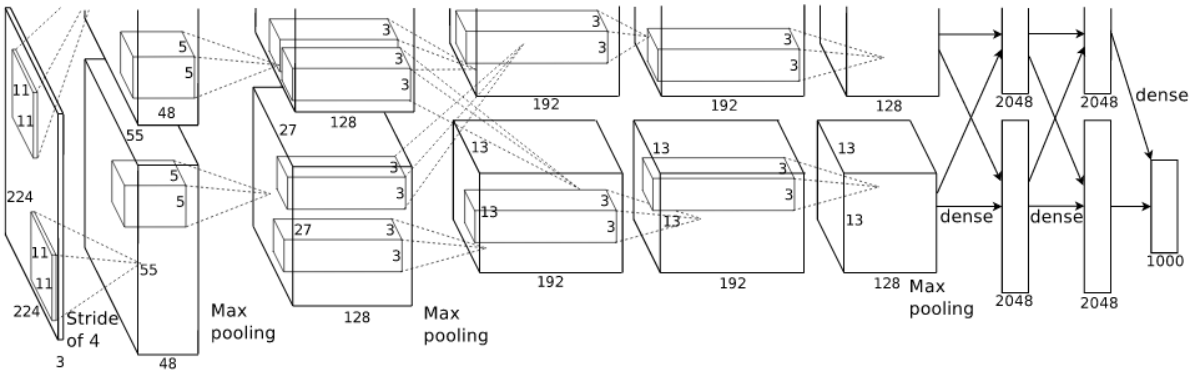


Figure 2.8: AlexNet Model [22].

[8] http://www.image-net.org/challenges/LSVRC/2014/

Other Models: There are many other popular architectures in the field of ConvNets like ZFNet [84], which was the winner of ImageNet competition 2013. Another one is GoogLeNet [24], which was the winner in that same competition in 2014. In addition, VGGNet [23] was the second ranked in the same competition in 2014 and they showed that the deeper a model the better its performance. Finally, ResNet [25] was the winner in the same competition in 2015.

2.4.3. Recurrent Networks

Many applications have sequences of inputs. For instance, a sequence of words is often displayed through sentences, where a one-hot vector (i.e. a vector of all zeros except for a single 1 at the index of the word in a fixed vocabulary) is depicted by each word. A connectivity pattern, which processes a sequence of vectors $\{x_1, \dots, x_T\}$ using a recurrence formula of the form $h_t = f_\theta(h_{t-1}, x_t)$ is referred to as a recurrent neural network RNN [88], where *f* is a function which will be explained later. Moreover, every time step uses the same parameters θ, through which we are able to process sequences with an arbitrary number of vectors [60].

As given in Figure 2.9, mathematically, RNN can be represented by $h_t = \mathrm{f}(Ux_t + Wh_{t-1})$ where h_t is the hidden state at time step *t*, *f* is a function of the input x_t weighted by the weight matrix *U* with the hidden state of the previous time weighted by the weight matrix and *W*. $O_t = f(Vh_t)$ is the output state and weighted by matrix *V* while *Y* is the actual value. Finally, *L* is the loss function, to compute the disagreement between *Y* and *O*.

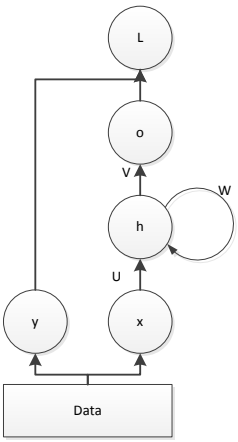


Figure 2.9: RNN architecture.

The main limitation of RNN is a problem known the vanishing and exploding gradient problem which can occur during the training [97-99]. Where the vanishing gradient problem occurs when the gradient tends to get smaller as we move backward through the hidden layers. This means that neurons in the earlier layers learn much more slowly than neurons in later layers. While the exploding gradient problem occurs when the gradient gets much larger in earlier layers.

Long Short-Term Memory: The limitations of the basic RNN can be addressed by designing the LSTM [100]. Figure 2.10 shows the LSTM model.

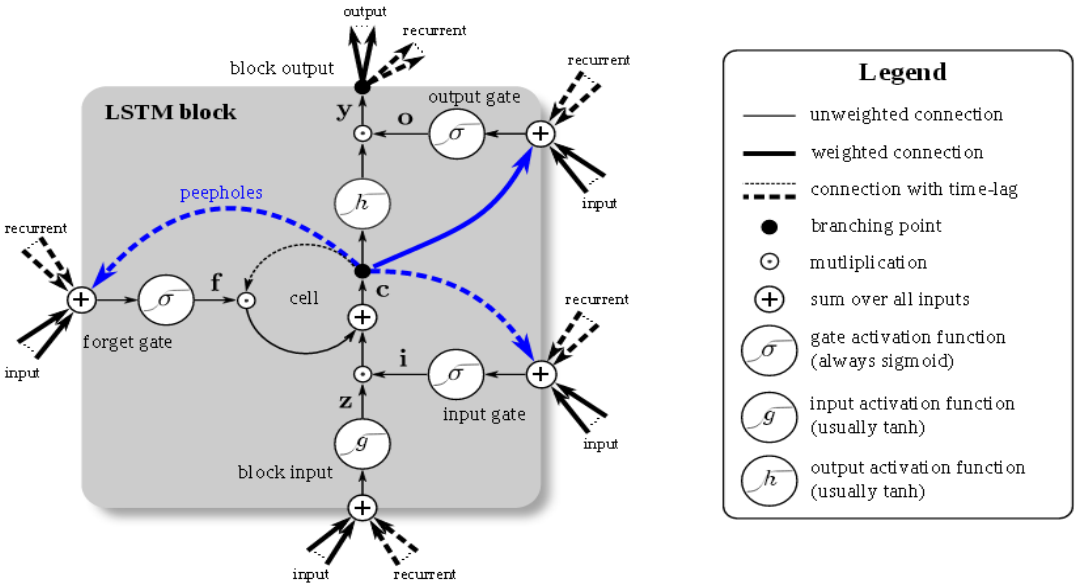


Figure 2.10: LSTM architecture [100].

The inputs x_t and h_{t-1} interact in a more computationally complex manner due to the nature of its recurrence formula and multiplicative interactions are involved in this mechanism. Moreover, additive interactions over time steps are used by the LSTM recurrence, which efficiently propagate gradients backwards in time [100]. Besides a hidden state vector h_t, a memory vector c_t is also maintained by the LSTM. By using explicit gating mechanisms, the LSTM can perform the functions, such as: select, write to, read from, or reset the cell at each time step. Figure 2.10 shows one neuron of a LSTM.

In Figure 2.10, the output can be given by $y^t = o^t \odot h(c^t)$ where t is the current time, o is the output of the output gate, c is the output of the cell state and Θ is the point-wise multiplication of two vectors. The output of the output gate is given by $o^t = \sigma(W_o x^t +$

$R_o y^{t-1} + P_o \odot c^t + b_o$) where x^t is the input at time *t*, $y^{t-1}$is the previous output, $R_o$is the peephole weight and *b* is a bias. The cell state can be given by $c^t = i^t \odot z^t + f^t \odot c^{t-1}$ where i^t is the output of the input gate at time *t*, $z^t$is the output of the input to the neuron and f^t is the output of the forget gate. The forget gate can be given by $f^t = \sigma(W_f x^t + R_f y^{t-1} + P_f \odot c^{t-1} + b_f)$. The input gate is given by $i^t = \sigma(W_i x^t + R_i y^{t-1} + P_i \odot c^{t-1} + b_i)$ and finally, the block input is given by $z^t = g(W_z x^t + R_z y^{t-1} + b_z)$[100].

End-To-End Memory Networks: The main goal of memory networks is offering memory to read and write to it in the long term. The main use of this kind of RNN is that it can read a story and then answer questions by extracting from the story. The other is as dialog agents. Figure 2.11 shows an End-to-End memory network, as presented in [101].

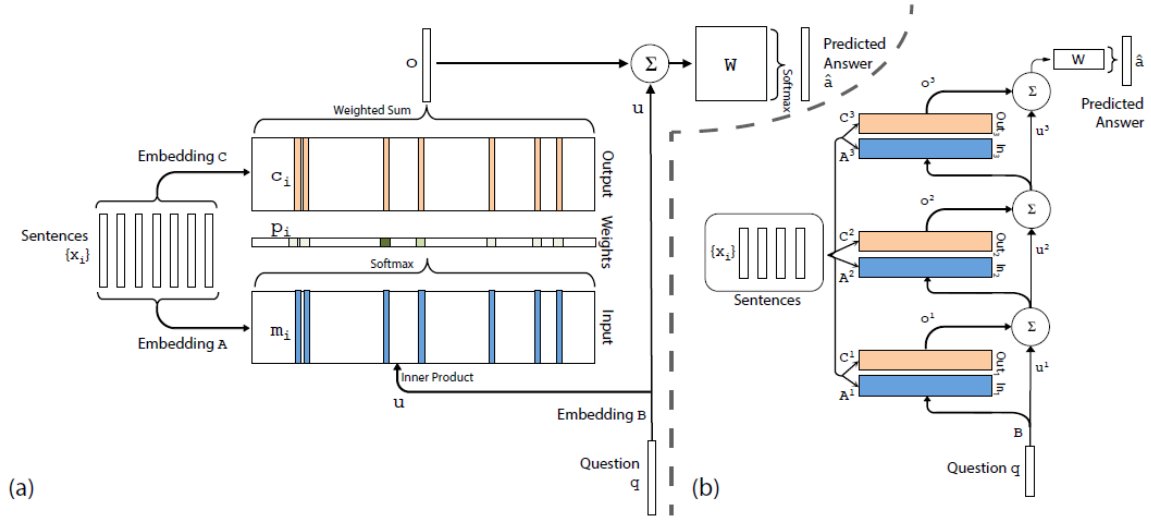


Figure 2.11: End-To-End Memory Networks. (a): A single layer. (b): A multiple layer.

In the Figure 2.11 (a), the single layer contains two parts, the memory module where the story is read, then saved in memory; and the controller module used to address and read between questions and the memory. Every input sentences x_i is embedded twice m_i and c_i. In a question and answering problem, the story will be embedded in the memory module and the question will be embedded in the controller module. Assume the story has three sentences followed by a question: John moved to garden, John went to kitchen, John drops apple. Where is John?

The representation of this story will be as follows:

First: the input sentences will be embedded to memory vectors in the memory module and the question in the story will be embedded to the controller as following

John moved to garden ➔ m_1 *and* c_1 *(*➔ *means embedded)*

John went to kitchen ➔ m_2 *and* c_2

John drops apple ➔ m_3 *and* c_3

So m_1,m_2 and m_3 will be saved in memory module (Embedding A) and c_1,c_2 and c_3 (Embedding C).

where is John ➔ u_1

Second: Apply the dot product between the memory vectors m_1,m_2,m_3 and controller vector followed by Softmax to get the attention weights such that:

$$p_i = Softmax(u^T m_i) = \exp(u^T m_i) \Big/ \sum_j \exp(u^T m_j)$$

Third: compute the weighted sum of the memory vectors using:

$$o = \sum_i p_i c_i = p_1 c_1 + p_2 c_2 + p_3 c_3$$

Finally: the output *o* adds it back to the controller where the actual output (*W*) is *kitchen* and back propagation is used to learn weights.

Figure 2.11 (b) shows multiple layers where it extends the model to handle *K* hop operations. In the previous example, the answer *kitchen* is found is the second sentence that means there is two hop operations. If the answer is found in the fourth sentence, then there are four hop operations.

2.4.4. Challenges of Training Neural Networks

Historically, neural networks have been considered to be hard to train, especially if the networks have more than one layer [102] for many reasons. The first reason is underfitting, in other words the networks cannot learn complex functions. The reason for this problem could be that there is not enough data or a vanishing gradient problem. To avoid this problem, researchers collect more data. The second reason, as aforementioned, is overfitting in the deep learning field. The reason for overfitting is that the number of parameters is high so the model learns the noise in the data, which leads to bad generalization. The following are the most recent popular techniques applied for deep learning in general in addition to the old techniques like regularization.

Dropout parameters: There are two popular types of dropout used for training the models to avoid overfitting.

Firstly, there is the dropout of the hidden neurons [103]. This type of dropout was introduced by Hinton and his group, contributed to winning the ImageNet competition in 2012 [104]. The idea behind this dropout technique is to remove hidden neurons stochastically by some probability. In other words, some hidden neurons are set randomly to zero by some probability. First, some hidden neurons are randomly set to zero with a given probability. Then, the same step is taken in the next layer and so on. Figure 2.12 (a) shows the neural networks after applying random dropout regularization.

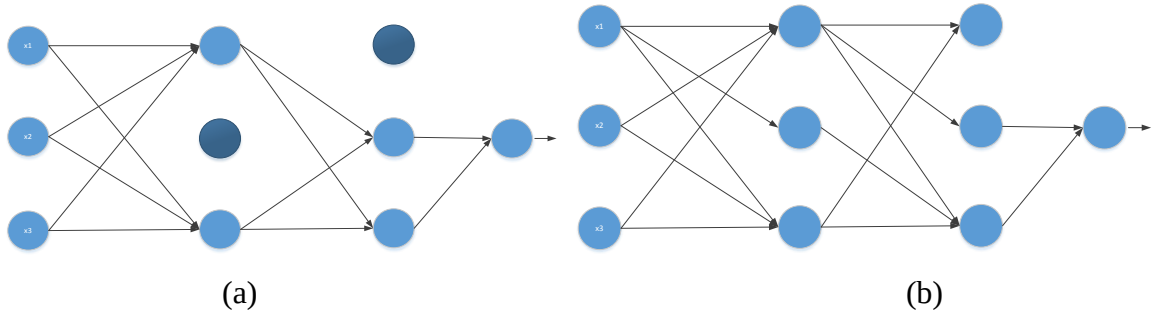


Figure 2.12: Neural networks (a) after dropout (b) after DropConnection.

Secondly, there is dropout on the connections (weights) and this is called DropConnection [105]. This uses the same idea as general dropout but instead of randomly setting some neurons according to a given probability, some connections between layers are removed randomly with given probability. Figure 2.12 (b) shows neural networks after DropConnection regularization.

Batch Normalization: To overcome the problem of internal covariate shift[9], a smaller learning rate and vanishing gradient, batch normalization[106] is used. Batch normalization is a transformation that is applied to the activation neuron over the mini batch input (m). Batch normalization can be given by the following steps, as cited by Ioffe & Szegedy [106];

1. The first step is to compute the mean and the variance of the mini batch as:

[9] Change in the input distribution leads to change in the learning system

$$\mu_B \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \text{ and } \sigma_B^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_B)^2$$

2. The second step involves normalizing the mini batch input as follows:

$$\hat{x}_i \leftarrow \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \varepsilon}}$$

3. Finally, the normalized mini batch is scaled using learnable parameters (γ, β):

$$y_i \leftarrow \gamma \hat{x}_i + \beta$$

2.5. Summary

This chapter covered the main concepts behind deep neural networks, as well as the principles relating supervised learning. Different learning optimization techniques and back propagation presented in the chapter. In addition, ConvNets and RNNs have present along with their architectures.

3. Literature Review

The design of a neural network typically begins by defining its topology, namely the number of layers, the number of neurons for each of the layers, and how the layers are connected. The network is then trained which involves experimentation, including determining suitable hyperparameters, such as the learning rate. As outlined in the introduction to this thesis, deep learning networks can become huge, and hence several methods have been developed for reducing their size. This chapter presents a survey of existing algorithms for pruning neural networks and gives an overview of the other methods of designing efficient the neural networks.

3.1. Background on Pruning Methods

There are many types of neural networks, such as feed forward neural networks (FNNs), convolution neural networks (ConvNets), recurrent neural networks (RNNs) and recursive neural networks (RecNN). An important practical question is: *What is the size of a FNN, ConvNets and RNN that leads to optimal performance?*

Theoretically Montufar et al. [107] and empirically He et al. [25], Simonyan & Zisserman [23], Szegedy et al. [24, 108] , Huang et al. [109] show that as a ConvNet gets deeper its performance gets better even though there might be increased redundancy [29]. Most of the networks are trained by leveraging high performance parallel architectures such as GPUs [22, 110], or distributed clusters [74]. These models have a huge number of weights and neurons, many of which might be unnecessary [29].

There is an extensive Literature on pruning neural networks (Reed [111] and Augasta & Kathirvalavakumar [112]). The literature can be divided in two periods:

- In late of 80s and 90s, the pruning methods focused on forward neural networks
- In the last three years, the pruning methods focus on pruning ConvNets.

The timeline of most related algorithms to our work are shown in Figure 3.1. The first mention of the importance of pruning neural networks dates to Kruschke [113] until the current days Wolfe et al [114].

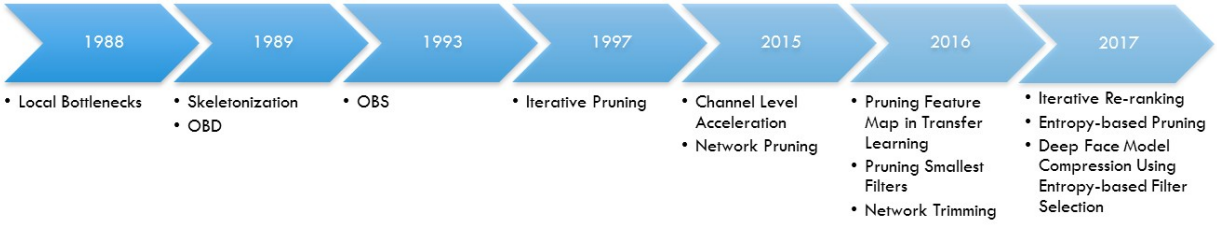


Figure 3.1: A timeline of related algorithms.

These algorithms can be classified based on the type of pruning that they perform which pruning the weights, neurons, or feature maps (as shown in the first column) and based on the approach of pruning (as shown in the first row) as be presented in Table 3-1.

	Magnitude	**Activation**	**First order**	**Second order**
Weights	Network pruning			OBD, OBS
Neurons	Local Bottlenecks	Iterative Pruning	Skeletonization, Iterative Re-ranking	Iterative Re-ranking
Feature maps	Pruning Smallest Filters	Channel Level Acceleration, Network Trimming, Pruning Feature Map in TL, Entropy-based Pruning, Deep Face Model Compression Using Entropy-based Filter Selection	Pruning Feature Map in TL	

Table 3-1: Summary of related work.

The following is the description of the methods based on the approach of pruning as they are discussed in the literature:

1. Direct methods [111] (also known as brute force pruning [114] or oracle pruning [115]) where, one by one, a weight, neuron or feature map are set to zero, the network's performance measured and then a decision is made to retain or remove the weight, neuron or feature map.
2. Regularization methods in which a regularization term *R(W)* is added to the loss function:

$$\frac{1}{N}\sum_{i=1}^{N} L_i + \frac{\lambda}{N} R(W)$$

Where $\lambda > 0$ is a regularization parameter that can be set to a value that reflects the weight of the regularization. There are many types of regularization [63-67], and two that have been widely used are the *L2* and *L1* norms:

L2 norm: $R(W) = \frac{1}{2}\sum_w w^2$

Where *L2* norm is known as weight decay and is the sum of the squares of all the weights in the network

L1 norm: $R(W) = \sum_w |w|$.

L1 norm is the sum of the absolute values of the weights:

3. Pruning based on magnitude [63, 116-118], pruning based on the magnitude of the weights is perhaps the simplest method. The motivation of using this method is that after training the deep neural networks with regularization, unimportant weights are pushed to zero. Hanson & Pratt [116] and Chauvin [63] add bias terms to the loss function to penalize the weights then the weights that are smaller than a predefined threshold are removed.
4. Activation methods [36, 114, 115, 119-121], removing neurons or feature maps based on their outputs.
5. First order pruning [122] exploiting information contained in the error gradient for better adapting neural network structure to the data and for improving its generalization. After computing back propagation, weights [122-124], neurons [36, 114] or feature maps [115] that have less impact on the error gradient are less important. Recently, Taylor approximation of the error can be simplified by eliminating second and higher order to estimate the pruned feature maps [115].

6. Second order pruning [122]. Second order information in the error can be exploited to prune insignificant weights [122]. Methods based on the second order partial derivative for modelling the error by using the Taylor series expansion to estimate the unimportant weights or neurons [125-128] where the third and higher order terms of are eliminated.

These methods have been shown to have different merits. The direct methods are of $O(NP^3)$, where *P* is the number weights, neurons or feature maps and *N* is the size of the training set, and hence are considered to be intractable [111, 114, 115] . Collins & Kohli [129] show that regularization (*L1* or *L2* norm) may not reduce the weights to zero in neural networks [130]. In addition, Gupta et al. [131] present that regularization method does not actually delete weights from the network, nor does it typically produce weights that are exactly zero. Weights that are not essential to the solution decay to zero and can be removed using pruning based magnitude [131]. Hassibi et al. [126] show that pruning based on magnitude can lead to pruning important weights. Srinivas & Babu [132] conclude that Taylor expansion methods have difficulty in pruning deep neural networks and Wolfe et al. [114] have recently shown that direct methods outperform the pruning method based on second order derivatives, which in turn are known to be better than the pruning method based on first order derivatives [114].

3.2. Related Work

The following are the related work to the proposed methods classified based on their goal which prune the weights, neurons, or feature maps. Followed by the other techniques which use different techniques to make deep neural networks more efficient.

3.2.1. Pruning Weights

Setting one or more connections between two neurons to zero and three different methods are related to the proposed methods:

3.2.1.1. Optimal Brain Damage

Optimal Brain Damage (OBD), a method developed by LeCun et al. [125], was one of the oldest methods for reducing the size of neural networks. OBD removes the weights that if

set to zero would have least effect on the training error. To measure the effect of changing weights, LeCun et al. [125] used a Taylor series approximation for the change in loss that would occur if the weights were perturbed. This analysis leads to the need to solve a Hessian matrix which can be computationally expensive. To reduce this computational cost, LeCun et al. [125] ignored the off-diagonal values.

For a loss function $L(x)$, the Taylor expansion evaluated at the point a is defined by [115, 122, 125-128]:

$$L(x) = L(a) + \frac{1}{1!}L'(a)(x-a) + \frac{1}{2!}L''(a)(x-a)^2 + \frac{1}{3!}L'''(a)(x-a)^3 + \cdots$$

Given the weight w where $\mathrm{w} \in \mathrm{R}^{\mathrm{K}}$ and K is the total number of the weights in the model then a small change on the weight vector w denoted by δw, will cause a change in the loss function denoted by δL

$$\delta L = L(w + \delta \mathrm{w}) - L(w)$$

LeCun et al. [125] use the Taylor expansion to derive the following equation for the change in loss when the weights are perturbed:

$$\delta L = \sum_i g_i \delta w_i + \frac{1}{2}\sum_i h_{ii}\delta w_i^2 + \frac{1}{2}\sum_{i \neq j} h_{ij}\delta w_i \delta w_j + O(||\delta W||^3) \qquad (3.1)$$

Where $g_i = \frac{\partial L}{\partial w_i}$ and $h_{ij} = \frac{\partial^2 L}{\partial w_i \partial w_j}$. The second order derivatives, h_{ij} can be presented in the form of a matrix H that is known as a Hessian matrix [125].

OBD makes some assumptions to the Equation 3.1.

First, computing the optimal H is computationally expensive [125, 132, 133] as it needs to calculate the second derivative of all the weights. Hence, they introduce diagonal approximation, which means there are no cross correlations between the perturbations of multiple weights. The off-diagonal terms in Equation 3.1 will set to zero (which is the third term in the Equation 3.1). In other words, OBD assumes that the change in loss, δL, is caused only by each weight individually.

Second, it assumes that the loss L is approximately quadratic and can be safely approximated by a second order Taylor Series. That is, the last term in Equation 3.1 is assumed to be zero.

Finally, OBD's equation assumes that the first term in Equation 3.1 can be set to zero given the network can be expected to be trained to reach a local minimum. After these assumptions Equation 3.1 is approximated by:

$$\delta L = \frac{1}{2}\sum_i h_{ii}\delta w_i^2 \qquad (3.2)$$

In Equation 3.2, δL shows how the loss changes with respect to perturbations of the weight vector. The δL relies on the diagonal terms of the Hessian matrix, given by:

$$h_{kk} = \sum_{ij} \frac{\partial^2 L}{\partial w_{ij}^2} = \sum_{ij} \frac{\partial^2 L}{\partial a_i^2} x_j^2$$

Where x_i is the state of neuron i, a_i its total input (weighted sum).

Thus, based on Equation 3.2, the extent to which a weight effects the loss (i.e., the saliency) is given by:

$$s_k = h_{kk} w_k^2 / 2$$

Where s_k is the saliency of the weights w_k

Finally, the weights are put in order and those with the lowest saliency are removed and the network retrained.

3.2.1.2. Optimal Brain Surgeon

Optimal Brain Surgeon (OBS) is a method due to Hassibi et al. [126-128] in which the Hessian matrix that results from the analysis for OBD is solved without making the assumption that the off-diagonal elements can be ignored. The other assumptions, such as ignoring higher order terms and the first term of Equation 3.1 are also assumed in OBS. That is, Equation 3.1 is rewritten to:

$$\delta L = \frac{1}{2}\delta w^T . H . \delta w \qquad (3.3)$$

Where $H = \frac{\delta^2 L}{\partial w^2}$.

Equation 3.3 leads to the following measure of saliency for a weight w_q [126-128] that is analogous to the one used for OBD:

$$L_q = \frac{1}{2}\frac{w_q^2}{[H^{-1}]_{qq}}$$

Both OBD and OBS are known to be very slow as they are based on computing the Hessian matrix [132].

3.2.1.3. Network Pruning

Network Pruning methods [134] remove weights that are below a user specified threshold value and then retrain the network. Before retraining, they improve dropout (as pruning already reduced the networks capacity, the retraining dropout ratio needs to be smaller [134, 135] giving the same training data set) by a ratio of the square root of the number of connections after pruning to the number of connections before pruning as given by:

$$D_r = D_0 \sqrt{\frac{C_{ir}}{C_{io}}}$$

Where D_r and D_0 is dropout after and before pruning respectively, C_{ir} and C_{io} are the number of links in layers after and before retraining respectively. To find the unnecessary weights, greedy method is used.

This approach to pruning is adopted for a number of different types of neural networks. For instance See et al. [136] and Narang et al. [137] also use thresholds to prune RNN and Han et al. [138, 139] use this method to compress ConvNets and LSTM respectively.

Network Pruning using thresholds is a simple to adopt, however, Srinivas & Babu [132] have found that it can result in the removal of weights that are important.

3.2.2. Pruning Neurons

Pruning neurons includes pruning all the connection from and into the neurons which is more efficient than pruning weights. There are four different methods for pruning neurons that are related to the proposed methods in the thesis.

3.2.2.1. Local Bottlenecks

Local Bottlenecks is a method in which the hidden neurons compete with each other to survive [113]. Magnitudes of vectors determine the degree to which a neuron affects the loss function. The gain is a new parameter inside the activation of a neuron to

indicate how much a neuron participates in representing the input. It needs special back propagation which is called back propagation with adaptive gains [113].

When the gain of a neuron is zero, it contributes only a bias term to upcoming layers and no error to back propagate and it is safe to be removed. In case two neurons have parallel or anti-parallel weight vectors, they are redundant and can be removed as well. The gain for neuron *i* from a neuron *j* (in the same layer) can be obtained from Equation 3.4 as:

$$\delta g_i^p = -\gamma \sum_{j \neq i} \langle \hat{w}_i^p, \hat{w}_j^p \rangle^2 . g_j^p \qquad (3.4)$$

Where γ is a small positive constant such as 0.001, $\hat{w}_i^s$ is the unit vector in the direction w_i^s, the operation $\langle ., . \rangle$ represents the inner product, and the superscript *p* is the pattern. If neuron *i* has weights parallel to those of neuron *j*, then the gain of each will decline in ratio to the gain of the other and the one with the lesser gain will be directed to zero faster. Since the gains should not be negative, this regulation can only decline them [111].

3.2.2.2. Skeletonization

Skeletonization is a method for pruning networks that was proposed by Mozer & Smolensky [36]. As with the other methods, they seek to assess the relevance, ρ_i, of a neuron by considering what happens if the neuron is removed:

They introduce a new concept, which is called attentional strength α, which is responsible for determining whether a neuron should be removed. When α_i, the attentional strength at neuron *i*, is equal to zero then neuron *i* has no effect on the rest of the model and when it is equal to 1 it works like a conventional neuron. That is the output of a neuron *j* is defined by $o_j = f(\sum_i w_{ji} \alpha_i o_i)$ where w_{ji} is the weight between neurons *i* and *j*, *f* is a nonlinear function, α_i are not actual parameters of the system but represent attentional strength.

The relevance of a neuron, ρ_i, is computed using:

$$\rho_i = L_{\alpha_i=0} - L_{\alpha_i=1} \approx -\frac{\partial L}{\partial \alpha_i}|_{\alpha_i=1}$$

where $L_{\alpha_i=0}$ is the loss when $\alpha_i = 0$ (i.e., the same setting the neuron to zero) and $L_{\alpha_i=1}$ is the loss when the $\alpha_i = 1$ (i.e., the neuron behaves normally).

Mozer & Smolensky [36] compute the derivative ρ_i during the training process with a procedure similar to back propagation but observed that the measure can change dramatically from one training cycle (*t*) to the next (*t+1*), and they therefore introduce the following weighted measure, $\widehat{\rho_i}$ which produces better estimates as:

$$\widehat{\rho_i}(t+1) = 0.8\widehat{\rho_i}(t) + 0.2\,\frac{\partial L(t)}{\partial \alpha_i}$$

It is worth observing that this method is based on first order derivatives and that other methods based on first-order derivatives have some limitations when compared to second order derivatives and the direct method [114].

3.2.2.3. Iterative Pruning

The Iterative Pruning algorithm, developed by Castellano et al. [140], removes a hidden neuron and then adjusts the weights of the network using a method known as the conjugate gradient preconditioned normal equation (CGPCNE) [141].

Castellano et al. [92] define two groups of neurons which they term as the projection and receptive fields. Given, a neural network $N = (V, E, w)$ where *V* is set of neurons, which is divided into V_I, V_H, V_O for input neurons, hidden neurons, and output neurons respectively, and $E \subseteq V \times V$ is the set of connections where each connection (*i,j*) is associated with weight $w_{ij} \in \Re$; for each neuron $i \in V$, they define the projection field P_i and receptive field R_i by:

$$P_i = [j \in V | (j,i) \in E]$$
$$R_i = [j \in V | (i,j) \in E]$$

In other words, they group the neurons that are connected to the removed neuron into two fields. First, the projection field which has the neurons that their output connect as input to the removed neuron. Second, the receptive field which holds the neurons that have one of their input's connections is the removed neuron's output.

The network is assumed to be trained on *M* training examples such that:

Given these definitions, the iterative pruning algorithm, selects a hidden neuron *h* as follows:

$$h = argmin_{h \in V_H} \sum_{i \in P_h} w_{hi}^2 ||\bar{y}_h||_2^2 \qquad (3.5)$$

$\bar{y}_h$ change in the output and P_h is projection hidden field. Then remove the neuron and adjust its weights in P_i and R_i using CGPCNE.

Castellano & Fanelli [142] use the same method for feature selection by applying the method to prune the input neurons instead of the hidden neurons. Fangju [143] uses the Generalized Inverse Matrix [144] algorithm instead of the CGPCNE algorithm for updating the remaining weights.

3.2.2.4. Iterative Re-ranking

Wolfe et al. [114] investigate two different methods for pruning neurons from neural networks with two hidden layers that are trained on the MNIST data. The methods are based on the change of the error using the Taylor series expansion up to the second order as given in 3.1, but the parameters here are neurons instead on weights. The change in the error of k^{th} neuron from Equation 3.1 given by:

$$\delta L_k^2 = -O_k . \frac{\partial L}{\partial O}|_{O_k} + 0.5 \, . O_k^2 . \frac{\partial^2 L}{\partial O^2}|_{O_k} \qquad (3.6)$$

Where L is the loss and O_k is the output at neuron k.

The first method is based on computing δL_k^2 using second order back propagation, then ranking this change and selecting all those below a fixed threshold.

The second method is based on a greedy algorithm. It computes δL_k^2 for each neuron and then prunes the one that has the least value. It then performs forward and back propagation to compute δL_k^2 to remove another neuron with the least value and repeats the process until some stopping criterion is met (such as the maximum number of neurons to remove, percentage scaling needed or maximum allowable accuracy [114]). This method is computationally more expensive but considers the dependencies the neurons might have on one another which would lead to a change in error contribution every time a dependent neuron is removed.

However, this method shows that the brute force pruning is the optimal method for pruning neural network and we believe pruning based MAB is the best to mimic the brute force pruning. This method is limited to special nonlinearity functions like *sigmoid* and *tanh* where the second order derivative can be computed. However, modern deep learning frameworks use the *ReLU* activation function which has approximately zero value at second order derivative.

3.2.3. Pruning Feature Maps

In ConvNets, pruning the feature map improves the inference time and reduces the size of the model. This section describes five methods for pruning feature maps.

3.2.3.1. Channel Level Acceleration

Polyak & Wolf [119] propose an activation-based feature map pruning method which removes the feature maps with weak activation patterns and their corresponding filters. The key idea in this approach is to consider the low variance in the outputs of feature maps as an indication that a feature map is weak. A feature map $X_{i+1,j} \in \mathbb{R}^{w_{i+1} \times h_{i+1}}$ is generated by applying a filter $F_{i,j} \in \mathbb{R}^{n_i \times k \times k}$ to feature maps of the previous layer $X_i \in \mathbb{R}^{n_i \times w_i \times h_i}$, for instance, $X_{i+1,j} = F_{i,j} \times X_i$. Given N randomly selected images $\{x_1^n\}_{n=1}^N$ from the training data set, the statistics of each feature map can be estimated with one epoch forward pass of the *N* sampled data [119].

$$\sigma_{var-l_2}(X_{i+1,j}) = var\left(\left\{\left\|X_{i+1,j}^n\right\|_2\right\}_{n=1}^N\right) \quad (3.7)$$

$\sigma_{var-l_2}(X_{i+1,j})$ is the contribution variance of channel criterion, which is motivated by the intuition that an unimportant feature map has almost similar outputs for the whole training data set and acts like an added bias [119]. Then this feature map (weak feature map) is pruned.

3.2.3.2. Network Trimming

Hu et al. [120] prune weak feature maps based on the mean zero activation instead of the variance in Polyak & Wolf [119] method. Hu et al. [96] define Average Percentage of Zeros (APoZ) to measure the percentage of zero activations of a neuron after the ReLU mapping. The APoZ of the c^{th} neuron in i^{th} layer is defined as:

$$APoZ\left(O_c^{(i)}\right) = \frac{\sum_k^N \sum_j f(O_{c,j}^{(i)}(k)=0)}{N \times M} \quad (3.8)$$

$O_c^{(i)}$ denotes the output of c^{th} channel in i^{th} layer. A given filter is drawn across the entire previous layer, moved one or more pixel(s) at a time. Each position results in an activation of the neuron and the output is collected in the feature map. Then, $O_{c,j}^{(i)}$ will be tested across the dimension of output feature map of $O_c^{(i)}$ where the function *f* is 1 or 0 and *N* denotes the total number of examples. In their experiments, the number of examples is chosen to be 50,000 images from ImageNet.

3.2.3.3. Entropy-Based Pruning

Instead of determing a weak feature map based on the mean [120] or the variacne [119] of its output, Luo & Wu [121] propose an entropy-based metric to evaluate the importance of each filter. In their filter pruning scenario, if a feature map contains less information, its corresponding filter is less important, thus could be pruned. To compute the entropy value of a particular feature map, Luo & Wu [121] first divide it into *m* different bins, and calculate the probability of each bin. Finally, the entropy can be calculated as follows:

$$H_j = -\sum_{i=1}^{m} p_i log\, p_i \qquad (3.9)$$

Where, p_i is the probability of bin *i*, H_j is the entropy of feature map *j*. Han et al. [145] experiment the same method to prune networks given more data sets.

3.2.3.4. Pruning Feature Map in Transfer Learning

Molchanov et al. [115] introduce a method for pruning filters from ConvNets that relies on the first order Taylor expansion of the absolute change in the loss function. In every pruning iteration, one feature map is removed then the model's parameters are adjusted given one training example. More precisely:

$$|\delta L(h_i)| = |L(\mathcal{D}, h_i = 0) - L(\mathcal{D}, h_i)|$$

Where $\delta L(h_i)$ is the change of loss function, $L(\mathcal{D}, h_i)$ the loss function before removing the feature map (set the output to zero) and $L(\mathcal{D}, h_i = 0)$ the loss function when the output is set to zero ($h_i = 0$).

According to the Taylor polynomial near $h_i = 0$:

$$L(\mathcal{D}, h_i = 0) = L(\mathcal{D}, h_i) - \frac{\partial L}{\partial h_i} h_i + R_1(h_i = 0)$$

The higher order (R_1) is eliminated as the widely-used ReLU activation function has almost zero output in second order partial derivative. Finally, Molchanov et al. [115] obtain the following measure for deciding which feature map, h_i should be removed:

$$|\delta L(h_i)| = \left|\frac{\partial L}{\partial h_i} h_i\right| \qquad (3.10)$$

A potential weakness of this method is that pruning a feature map based on one example (greedy method) is not resistant to noise.

3.2.3.5. Pruning Smallest Filters

Hao Li et al. [146] remove the filters that have the smallest absolute sum among the filters in a convolution layer in ConvNet to reduce *FLOPS* (floating-point operations). Let $n_i$denote the number of input feature maps for the i^{th} convolutional layer, h_i is the height of the input feature maps and w_i is the width. $x_i \in \mathbb{R}^{n_i \times h_i \times w_i}$ are the input feature maps and $x_{i+1} \in \mathbb{R}^{n_{i+1} \times h_{i+1} \times w_{i+1}}$ is the output. This is achieved by applying n_{i+1}3D filters $\mathcal{F}_{i,j} \in \mathbb{R}^{n_i \times k \times k}$ on n_i 2D kernels $K \in \mathbb{R}^{k \times k}$. All filters together, constitute a matrix $\mathcal{F}_i \in \mathbb{R}^{n_i \times n_{i+1} \times k \times k}$. The operations of the convolutional layer are $n_{i+1} \times n_i \times k^2 \times h_{i+1} \times w_{i+1}$. Finally, remove filter $\mathcal{F}_{i,j}$ and its corresponding feature map $x_{i+1,j}$ [146].

Finally, pruning based on magnitude where this method laid down is simple but has drawback of eliminated effective weighs [115, 126-128].

3.3. Summary of other Methods for Pruning

There are many other methods that aim to reduce the size of deep neural networks or speed up the inference time. This subsection presents some bibliographic remarks about these other methods.

Srinivas & Babu [132] propose a method that prunes a neuron if it has similar weights with another neuron in the same layer and then adjusts the remaining weights. In addition, the following methods are orthogonal with the methods proposed in the thesis and can be used with the proposed methods [138]. Collins & Kohli [129] develop a method that investigates

the use of sparsity inducing regularizes to reduce the number of non-zero weights during training of Convolution Neural Networks. Chen et al. [147] introduce HashedNets to compress the deep model. HashedNets uses a hash function to reduce model size by randomly grouping network connections into hash buckets uniformly, such that connections in a hash bucket use a single parameter value. Chen et al. [148] develop FreshNets that converts weights to the frequency domain then use HashedNets to randomly group frequency parameters into hash buckets. Sainath et al. [149] use a low rank approximation to fully connect layers at training mode to reduce the number of parameters. They replace the fully connected layer with a linear layer that has a small number of hidden neurons. Xue et al. [150] use low-rank factorizations with singular value decomposition after training the model. Gong et al. [151] use vector quantization to compress the network.

Denton et al. [152] use singular value decomposition to reduce the size of pretrained ConvNets. Iandola et al. [153] develop SqueezeNet, which aims to achieve the same performance as AlexNet but with a smaller model to reduce the training time in the ImageNet data set.

To speed up the deployment time for the ConvNets, Denton et al. [152] exploit the linear structure of ConvNets and compress each layer individually by slightly deteriorating the performance of the original model. Jaderberg et al. [154] introduce exploiting low-rank decompositions of convolutional tensors to speed up the evaluation of ConvNets. Zhang et al. [155] develop networks with a set of low-rank filters in each layer. This was built after training a network without this constraint, where the simpler network was selected to approximate the original full rank network.

To reduce the time of training ConvNets, Hongsheng et al. [156] develop an algorithm using the sparse convolution method to perform learning of ConvNets for pixel wise classification of images. Lebedev et al. [157] decrease deployment time by using cp decomposition to compress a 4d convolution filter and decomposing it to many layers of low complexity. Vincent et al. [158] present a linear algebraic trick for computing both the value and the gradient update for a loss function that compares a very high-dimensional target with a (dense) output prediction. Mathieu et al. [159], Highlander & Rodriguez [160], Rippel et al. [161] and Pratt et al. [162] use Fourier transform for reducing the training computation time on ConvNets. This result can be used to quickly compute convolutions in the Fourier

domain, since an elementwise product is much less computationally intensive than a convolution [159].

3.4. Summary

This chapter has presented several pruning methods. These methods were categorised into several groups based the way of pruning.

1. Direct methods where, one by one, a weight, neuron or feature map are set to zero, the network's performance measured and then a decision is made to retain or remove the weight, neuron or feature map.
2. Regularization methods in which a regularization term is added to the loss function.
3. Pruning based on magnitude, which is based on the magnitude of the weights and is perhaps the simplest method. The motivation of using this method is that after training the deep neural networks with regularization, unimportant weights are pushed towards zero.
4. Activation methods, which remove neurons or feature maps based on their outputs.
5. First and second order pruning, which exploit information contained in the error gradient for better adapting neural network structure to the data and for improving generalization.

Pruning can be applied to reduce the number of weights, neurons or feature maps. Chapter 5,6 and 7 uses some of these methods to compare the results relative to the new MAB based methods developed in this thesis.

4. Multi-Armed Bandit

As motivated in the previous chapters, pruning neural networks involves a trade-off between the amount of pruning and the accuracy. Historically, Arrow et al. [163] was among the first to recognize the importance of developing a theory that supported decision making for the important trade-off between exploration and exploitation. This was followed by the seminal work of Lai & Robbins, [43, 164] in which they proved a lower bound for the regret of the finite-armed, multi-armed bandit problem. Since then, there has been significant research on developing algorithms that aim to achieve the lower bound.

The term multi-armed bandits refer to a framework that is based on modelling a gambler who faces a collection of slot machines and needs to select which machines to play in order to maximize the returns. Prior to each pull, the gambler will know the expected return or payoff, based on the previous history of pulls and will be able to use this to decide whether to exploit the best arm or explore other arms with the hope of gaining a greater reward.

This chapter summarises multi-arm bandit algorithms. Section 4.1 introduces the notation, Section 4.2 describes the algorithms and readers are referred to [165-167] for more complete and comprehensive accounts of the theory associated with the convergence properties of multi-arm bandits [165].

4.1. Notation

This section introduces the notation which will be used with multi-armed bandit in the rest of the thesis. The notation used is based on Burtini et al.[165] and Galichet [168]:

- K denotes the number of arms, machines, or options $K \in N^+$ in other word $K = \{2,3, \dots\}$;
- v_i denotes the (unknown) bounded reward distribution associated with the i^{th} arm;
- μ_i denotes the expectation of v_i ($\mu_i = \mathbb{E}[v_i]$);
- μ^* is the maximum expectation taken over all arms ($\mu^* = \max_{i=1\dots K} \mu_i$);
- δ_i is the optimality gap of the i^{th} arm ($\delta_i = \mu^* - \mu_i$);
- T denotes the time horizon ($T \in N^+$), which might be finite or infinite;
- t denotes the current time step;
- a_t is the arm selected at time t;
- $X_{i,t}$ is the t^{th} selected reward drawn from distribution v_i;
- $X_{a_t,t}$ is the reward obtained at time t. Every arm has its own unknown reward at time t so when the arm is selected, for example arm a_t then $\mathrm{X}_{i,t} = \mathrm{X}_{a_t,t}$ but we do not know the reward for the other arms;
- n_i is the number of times the i^{th} arm has been selected up to time t ($n_i = \sum_{k=1}^{t} \mathbb{I}_{a_{t=i}}$) where $\mathbb{I}_{a_{t=i}} = 1$when a_t is chosen otherwise is zero (it counts the number of times arm a_t is chosen during the time t).

The goal of multi-armed bandits is to maximise the reward over the time horizon. The maximum cumulative reward (gain or payoff) gathered along time H is defined by $\sum_{t=1}^{H} \mathrm{X}_{a_t,t}$. Maximizing the agent's total reward is equivalent to minimizing its total regret compared to the oracle (optimal) strategy which define by $\max_{i\in\{1\dots K\}} \sum_{t=1}^{H} X_{i,t}$ Then, cumulative regret R of the agent at time t is defined by

$$R = \max_{i\in\{1\dots K\}} \sum_{t=1}^{H} X_{i,t} - \sum_{t=1}^{H} X_{a_t,t} \qquad (4.1)$$

where the goal is minimizing the regret.

There are three broad categories of multi-armed bandits: (i) problems where the aim is to pull one arm at a time and maximize the total reward given a number of pulls and the world is stationary; (ii) adversarial problems where the goal is to play one arm at a time and maximize the total reward given a number of pulls and given the world is not stationary (iii) multi-play problems where the aim is to pull multiple arms at a time and maximize the total

reward given a number of pulls and the world is stationary. The following subsections describe algorithms in these categories.

4.2. Sequential Multi-Armed Bandits

There are five types of multi-armed bandit algorithms that have been proposed: random explorations [169], optimistic explorations [170], and Bayesian algorithms [171, 172]. The following subsections describe and present the algorithms that are utilized in this thesis.

4.2.1. Random Explorations

In random explorations [169], arms are pulled randomly, expected returns calculated and a strategy for deciding when to exploit the best arm or explore other arms is employed.

4.2.1.1. Random Selection Algorithm

In this technique, the next arm a_{t+1} is chosen randomly from the arms space. The main drawback of this method is that there is no guarantee the arm being chosen is the right arm that needs to be play.

4.2.1.2. Greedy Algorithm

The simplest approach to the MAB problem is to select arms randomly for a number of times (exploration), compute the average rewards and then select the best arm repeatedly (exploitation):

$$a_{t+1} = arg \max_{i \in \{1 \dots K\}} [\mu_i] \qquad (4.2)$$

Where μ_i is the empirical mean reward of i^{th} arm and time *t*.
One drawback of this method is it could lead to a suboptimal arm being selected. In other words, there might be a better arm if the initial random exploration has not been long enough.

4.2.1.3. Epsilon-Greedy Algorithm

One common random exploration algorithm, known as ε-greedy [173-175] pulls (i.e. exploits) the current best arm with a probability $1-\varepsilon$, and otherwise pull another arm randomly (i.e., explore). More formally, the Epsilon-greedy algorithm selects the next arm a_{t+1} as follows:

$$a_{t+1} = \begin{cases} arg\,max[\mu_t(1), \mu_t(2), \dots, \mu_t(k)] , & with\ probability\ 1-\varepsilon \\ select\ an\ arm\ randomly\ from\ \{1..k\}, & with\ probability\ \varepsilon \end{cases} \quad (4.3)$$

Where there are *k* arms, and $\mu_t(i)$ denotes the current average reward.
Selecting a suitable ε for this algorithm can be challenging. where if ε is large then, the algorithm will waste many playing pulling random arms without gaining much while if ε is too small, then the learning will be slow [165]. Hence some authors have proposed a strategy of decaying ε over time [175]. For example, White [176] proposes decaying ε by $\frac{1}{\log(t+\varphi)}$ where φ is very small number and t is the round or number of plays to date.

4.2.1.4. Win-Stay, Lose-Shift Algorithm

Another technique, known as the Win–Stay, Lose–Shift (WSLS) heuristic is recognized as one of the most simplified models with which bandit problem decision-making can be done [177, 178]. The WSLS algorithm is based on pursuit methods [55] and changes the probability of the choosing an arm over time depending on whether it is selected or not in the current round. If the current arm is selected (i.e., wins), then it makes the probability stronger otherwise makes it weaker. More formally, let $P_t(a)$ be the probability of choosing arm a at time *t*, then the update equations are:

$$P_{t+1}(a) = \begin{cases} P_t(a) + \beta(1 - P_t(a)), & if\ a\ is\ winning \\ P_t(a) - \beta P_t(a), & otherwise \end{cases} \quad (4.4)$$

Where β is a scaling parameter for rewarding the winner or penalizing the loser.

4.2.1.5. Softmax Algorithm

The Softmax algorithm [174] uses the Gibbs (Boltzmann) distribution to estimate the probability for each arm based on the rewards for each arm. Arms that have a higher expected payoff will have a higher probability to be selected and P_i is given by:

$$P_i = exp^{\frac{\mu_i}{\tau}} / \sum_{k=1}^{K} exp\frac{\mu_k}{\tau} \quad (4.5)$$

Where $\tau \in \mathbb{R}^+$ is temperature and experiments are carried out to discover the value of τ. When τ is large, the model works like random selection and when it is small, it gives greater priority to those arms that have a higher mean value. Like, Epsilon-greedy, τ can be constant or decay over time.

4.2.2. Optimistic Explorations

As mentioned above, prior to pulling the next arm, the gambler will know the expected reward for each arm based on its history of previous lever pulls. A simple approach to selecting the next arm is to use the arm with the largest reward. However, this ignores the fact that the early estimates of the reward may be inaccurate. Thus, the main idea for optimistic exploration is to maintain confidence bounds on the expected rewards and to select the arm with the largest upper bound, ensuring there is sufficient exploration at the start, but also maximize exploitation given that the bounds tighten as the number of lever pulls increase. Multi-armed bandit methods that adopt this optimistic approach are known as Upper Confidence Bound (UCB) algorithms [164, 175], More formally, UCB algorithms aim to select the next arm, a_t as follows:

$$a_{t+1} = \underset{i \in \{1 \dots K\}}{argmax}(\mu_i + P_{f_i}) \quad (4.6)$$

Where μ_i is the expected reward for arm a_i and $P_{f_i}$is a padding function that is used to provide an upper bound for the reward for the arm.

4.2.2.1. UCB1 Algorithm

One of the earliest and most widely cited UCB algorithm is known as UCB1, which uses the following selection function:

$$a_{t+1} = \underset{i\in\{1\dots K\}}{argmax}\,(\mu_i + \sqrt{\frac{2log(t)}{n_i}}) \qquad (4.7)$$

Where n_i is the number of times arm *a* has been chosen and *t* is the total number of rounds. UCB1 begins by playing each arm once to create an initial estimate. Then, for each iteration *t*, arm *a* is selected using Equation 4.7. Initially, when arms have only been pulled a few times, the padding function in Equation 4.7 allows exploration, but as the number of rounds, *t*, increases and the number of times arms are played increases, the padding function reduces, leading to greater exploitation of the arm that returns the largest reward

4.2.2.2. KL-UCB Algorithm

KL-UCB [179] presents an alternative approach where the padding function is derived from the Kullback-Leibler (KL) divergence measure, leading to a selection function where the next arm to pull is given by:

$$a_{t+1} = \underset{i\in\{1\dots K\}}{\text{argmax}}\,(Q(i))$$

$$Q(i) = \max_q \left\{q \in [\mu_i, 1] : d(\mu_i, q) \leq \frac{log(t)+c\,log(\,log(t))}{n_i}\right\} \qquad (4.8)$$

Where *d* is the Kullback-Leibler divergence measure and *c* is constant. Kullback & Leibler [180], Garivier & Cappé [181] defines the Kullback-Leibler divergence with the Bernoulli distribution $d(p,q)$ as:

$$d(p,q) = plog\frac{p}{q} + (1-p)log(\frac{1-p}{1-q}) \qquad (4.9)$$

Where, $0log0 = 0$, $0log\frac{0}{0} = 0$ and $xlog\frac{x}{0} = +\infty$ for x>0.

4.2.3. Bayesian Bandits

In the Bayesian approach, the reward from each arm is represented by a probability distribution that is updated in a Bayesian fashion. If $P(R)$ is the prior probability of a reward, then the goal is to compute a posterior distribution $P(R|h_t)$ where h_t is the history of rewards and actions. Two different Bayesian bandit algorithms, namely Thompson Sampling and BayesUCB are described below.

4.2.3.1. Thompson Sampling Algorithm

One of the first algorithms that adopted a Bayesian approach was Thompson Sampling [41]. Given s_a, the number of times an arm results in a reward and, f_a, the number of times an arm fails to deliver a reward, the probability distribution for the arm is defined by the beta distribution [171, 172]:

$$P(x) = \frac{(1-x)^{\beta-1}x^{\alpha-1}}{B(\alpha,\beta)}$$

Where α is set to $s_a + 1$ and β is set to $f_a + 1$.

4.2.3.2. BayesUCB Algorithm

In a more recent development that uses a Bayesian approach, Kaufmann et al. [182] propose an algorithm BayesUCB, in which the quantiles of a distribution are estimated to increasingly tight bounds and used to determine the next step:

$$a_{t+1} = arg\max_{q} q_i(t) = Q(1-\frac{1}{t}, \lambda_i^{t-1}) \qquad (4.10)$$

Where Q is a quantile function for a distribution λ at the α level and is defined by:

$$\mathrm{Q}(\alpha,\lambda) \text{ such that } \mathrm{P}(\mathrm{X} \leq \mathrm{Q}(\alpha,\lambda)) = \alpha$$

4.2.4. Adversarial Bandits

Another form of the multi-armed bandit problem is called the adversarial bandit. In this form, at each iteration an agent chooses an arm and an adversary simultaneously chooses the reward structure for each arm. This is one of the strongest generalizations of the bandit problem [165] as it removes all assumptions of the distribution and a solution to the adversarial bandit problem is a generalized solution to the more specific bandit problems [165][10]. There are two kinds of adversarial bandit algorithms studied in the thesis and they are explained in the following subsections.

4.2.4.1. Hedge Algorithm

There are different variations of the Hedge algorithm [183] in the literature [184-186] and the version used in this thesis focuses on maximising rewards, and hence is presented in this section.

$$P_t = \frac{w_i(t)}{\sum_{j=1}^{n} w_j(t)} \quad (4.11)$$

$$w_i(t+1) = w_i(t)(1+\epsilon)^{\rho_i(t)} \quad (4.12)$$

The arm is chosen with probability proportional to the weights P_t, $\rho_i(t)$ is the current reward of the chosen arm and ϵ is very small number.

4.2.4.2. EXP3 Algorithm

The EXP3 (the exponential-weight algorithm for exploration and exploitation) algorithm [170, 175, 187, 188] is based on the Hedge algorithm.

The parameter γ is called the exploration rate and sets how much the algorithm will explore the action space. Where $\gamma \in [0, 1]$ but the standard setting is $\gamma = 0.1$ [170, 189] which means

[10] https://en.wikipedia.org/wiki/Multi-armed_bandit#cite_ref-40

10% of exploration. After the arm a is played and the reward ρ_a is given to the arm, its weight is updated using:

$$w_{a,t} = w_{a,t-1}.e^{\gamma.\frac{\rho_a}{p_{a,t}.K}} \quad (4.13)$$

In this expression, the $w_{a,t}$ is known as the arm a with particular weight and *t* is the time and *P* is the selection criteria. In each iteration, the probability of each specific arm to play next is given by:

$$P_{a,t} = (1-\gamma)\frac{w_{i,t}}{\sum_{i=0}^{K} w_{j,t}} + \gamma.\frac{1}{K} \quad (4.14)$$

4.2.5. Bandits with Multiple Plays

A new kind of problems is bandits with multiple plays where at each time *t* the policy chooses *m* arms. Examples applications where multiple play bandits can be useful include recommender systems and online advertising environments where several options need to be presented at the same time such as showing numerous advertisements on a single page or offering multiple product suggestions.

4.2.5.1. Thompson Sampling and Multiple Play Algorithm

An extended version of Thompson Sampling with binary rewards known as MP-TS (multiple play Thompson Sampling) is introduced by Komiyama et al. [190]. The adopt Thompson Sampling in which the sampling process provides the top *m* arms in each iteration instead of top 1.

An example application of multiple play bandits is presented by Burtini et al. [165], who use it to decide which products to display in an online advertising environment.

4.2.5.2. UCB1 and Multiple Play Algorithm

We extend the version of UCB1 to play multiple *k* arms at the same time in one play time and it is called MP-UCB1 (influenced by naming MP-TS). Instead of choosing the max arm in Equation 4.7, the policy will choose the best *m* arms from the list of all *K* arms.

4.3. Summary

This chapter has presented several multi-arm bandit algorithms. These algorithms were categorised into five categories based the way of exploring. Table 4-1 summarises the main algorithms and Their main characteristics are:

- Random exploration: This part includes some greedy, Softmax and WSLS MAB algorithms. After introducing a greedy method, some methods achieving an exploitation / exploration trade-off are presented and discussed.
- Optimistic: This section is devoted to the presentation of optimistic algorithms, which proceed by maintaining a confidence region for each arm expected payoff.
- Bayesian algorithms include the Thompson Sampling algorithms, introduced by Thompson [41] simultaneously to the Multi-Armed Bandit problem. They have since been extensively studied [182].
- Adversarial bandit. When the reward over the arm is not stochastic and changed over the time then this method works by maintaining a list of weights for each arm to perform. Using these weights, it decides randomly which arm to take next and increases/decreases the relevant weights when a payoff is good or bad. EXP3 and Hedge algorithm were discussed to represent the adversarial bandit.
- Multi-Play. Instead of playing one arm at one trail, these algorithms play multiple arms at one time in one trail. In this chapter, two algorithms were discussed.

Chapter 5,6,7 and 8 utilises these multi-armed bandits for developing algorithms for pruning neural networks.

Algorithm	Environment	Explore New Arm	No. of ARMS	Regret
ε greedy (Decay)	Stochastic	Random	Single	Bound
Softmax (Decay)	Stochastic	Probability	Single	Bound
WSLS	Stochastic	Probability	Single	Bound
UCB1	Stochastic	Optimistic	Single	Bound
KL-UCB	Stochastic	Optimistic	Single	Bound
Bayesian UCB	Stochastic	Bayesian	Single	binary
Thompson Sampling	Stochastic	Bayesian	Single	binary
Hedge	Adversarial	Probability	Single	Normal
EXP3	Adversarial	Random/ Probability	Single	Bound
MP-TS	Stochastic	Bayesian	Multiple	Binary
MP-UCB1	Stochastic	Optimistic	Multiple	Bound

Table 4-1: A comparison between multi-armed bandit algorithms.

5. Multi-Armed Bandit for Pruning Weights

This chapter introduces seven different MAB pruning algorithms to prune the weights of trained neural networks. The algorithms are implemented and their performance compared with the four existing algorithms: OBD, OBS, Random Pruning and Network Pruning. The comparison is based on developing neural networks for a selection of data sets from the UCI repository followed by application of the different pruning methods. The chapter is organized as follows: Section 5.1 presents the new MAB based pruning algorithms together with illustrative examples, Section 5.2 presents the results of an empirical evaluation, and Section 5.3 summarizes the chapter.

5.1. Architecture of MAB Pruning method

Figure 5.1 illustrates the key idea behind using MABs for pruning deep networks. A deep network is depicted at the top of the figure with numerous weights. Each weight is considered as a single arm, and when an arm is pulled (weight set to zero), it results in a reward. The reward is defined as the difference between the performance of the network before and after removing the weight based on applying the network on a random sample of the data. The arm (weight) selected together with the improvement become part of the history which is then used to select the next weight and the process repeated for a fixed number of rounds.

The reward function used varies depending on the type of bandit algorithm used. For UCB1, Epsilon-Greedy, KL-UCB and WSLS the reward is computed by first calculating the difference in the loss function ΔL and then computing the reward, $X_{a_t,t}$:

$$\delta L = L(D|W) - L(D|W')$$

$$X_{a_t,t} = \frac{max(0, Threshold + \delta L)}{constant} \quad (5.1)$$

Where L is the loss when the network is applied on example of the data D, W denotes the weights, and W' the weights after pruning. This definition of threshold determines how much loss in the performance is allowed when pruning any weight. For example, suppose pruning results in a slightly worse performance, resulting in δL = -0.05(say), then a threshold of 0.1, would still result in a reward.

For example, if the performance of the model before and after pruning is the same then, $max(0, Threshold + \delta L) = max(0, Threshold)$. That is, if the performance does not change, and the Threshold is positive, the reward will be $Threshold$.

The divisor Constant is defined in a way that ensures that the reward is bounded between zero and one. One possible way for choosing it is

$$\text{Constant} = \max expected\ performance\ - (The\ cunrrent\ performance + Threshold)$$

On the other hands, Thompson Sampling and BayesUCB algorithms assume Bernoulli rewards[11], and hence in this work the reward is 1 if δL is larger than zero and zero otherwise.

[11] Rewards 0 and 1 are referred to as a success and a failure, respectively

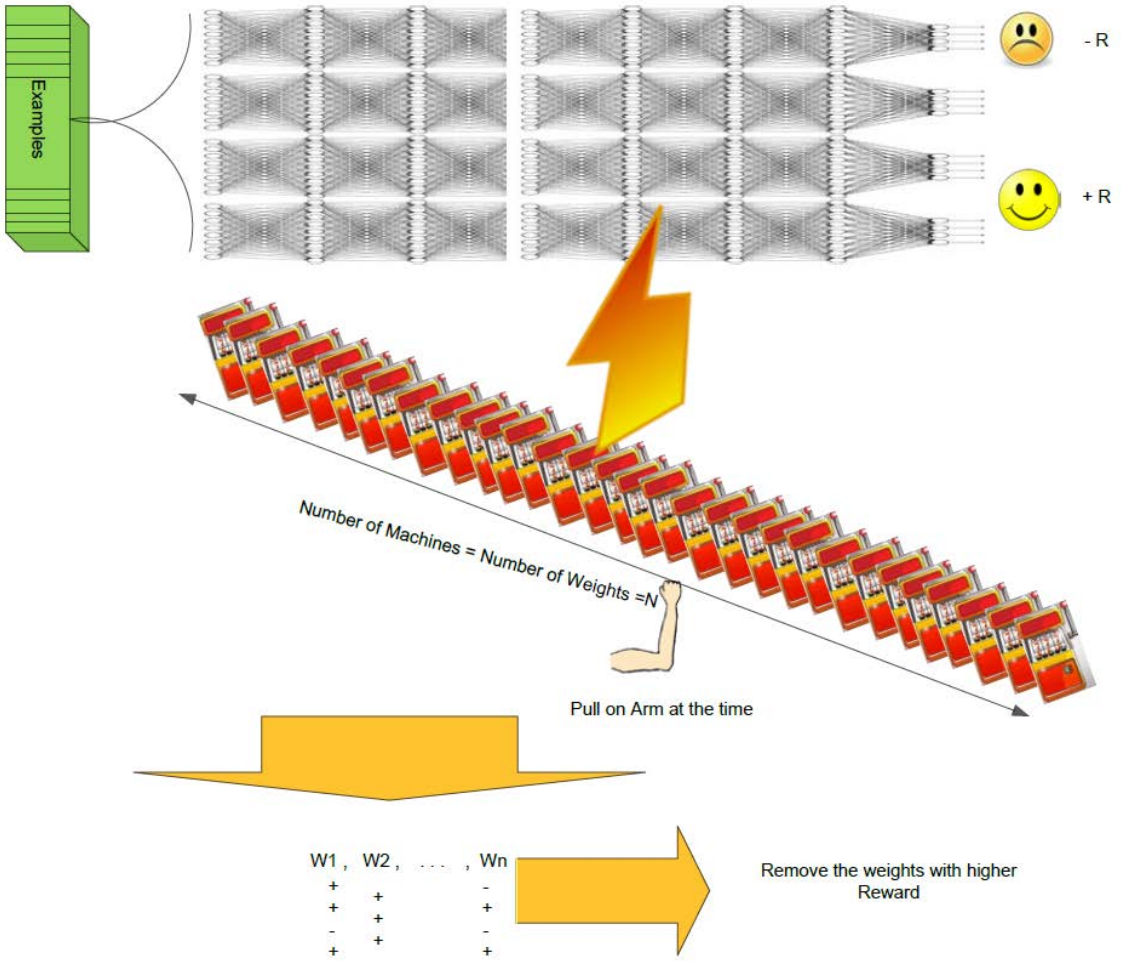


Figure 5.1: Block diagram shows the MAB to prune the weights.

The MAB based pruning algorithm, reflecting the process outlined earlier, is presented in 5.2, where Step 3 would involve invoking the specific MAB algorithm, such as UCB, KL-UCB, BayesUCB and Thompson Sampling that were described in Chapter 4 and will discuss in this chapter.

Algorithm 5.1 MAB Algorithm for pruning the weights

INPUT: Time horizon T, Trained network, Input layer l to be pruned.
OUTPUT: Pruned network
Let w_{ji} be the weight connection between two neurons j and i
Let $X_{ji,t}$ be the current reward of this arm/weight which equivalent to $X_{a_t,t}$ where the arm a is denoted by the connection between the neurons j and i.
Let μ_{ji} be the cumulative average reward of this arm/weight, initialized to zero
Let D the random sample/example from training data set
Let $L(D|W)$ The loss function before pruning the weight
Let $L(D|W')$ The loss function after pruning the weight
Let Threshold how much loss or gain in the performance is allowed
Let K is the number of weights in the chosen layer which equivalent to JI where J is the total number of neurons in previous layer and I is the total number of neurons in the current layer.
Let n_{ji} is the total current play time for the w_{ji}.
Let S_{ji} is the success and F_{ji} is the failure, both initialized to zero

1. for t=1 to T do /* start playing */
2. D = random example of the training data
   if any $n_{ji} = 0$ then choose the current index j, i /* Pull each weight at least once */
3. else Call the relevant MAB algorithm, returning the index of the selected weight:
   $j, i = MAB(* arg)$
4. Perform forward propagation on D to compute $L(D|W)$
5. Hold the value of the selected weight, $temp = w_{ji}$
6. Set the weight to zero $w_{ji} = 0$
7. Perform forward propagation on D to compute $L(D|W')$
8. Set the value of the weight to previous value, $w_{ji} = temp$
9. $\delta L = L(D|W) - L(D|W')$
10. $X_{ji,t} = REWARD(\delta L)$
11. Update the cumulative average reward of the current arm
    $\mu_{ji,t+1} = (n_{ji,t} - 1)/n_{ji} * \mu_{ji,t} + 1/n_{ji} * X_{ji,t}$
12. end for
13. $PrunedModel = PrunedFunction(model, \mu)$
14. end main program

15. $Function\ PrunedFunction(model, rewards)$

16. Set to zero the weights that have most rewards
17. return PrunedModel
18. end Function

19. $Function\ REWARD(\delta L)$

20. if (bounded reward) then /* For example, reward for UCB1, */
21. $Reward += max(0, \delta L + Threshold)\ /Constant$
22. else /* Reward for Thompson Sampling and BayesUCB */
23. if $\delta L < 0$ then reward=0, $F_{ji,t+1} = F_{ji,t} + 1$
24. else reward=1, $S_{ji,t+1} = S_{ji,t} + 1$
25. end if
26. end if
27. end Function

Figure 5.2: The generic algorithm of a MAB pruning the weighs.

Example: To illustrate the idea behind different MAB algorithms for pruning weights, a small neural network model with two inputs, one hidden layer with eight neurons, and two outputs was created and trained using synthetic data[12] consisting of the 1000 examples shown in Figure 5.3. The network was trained with 100 epochs and logistic loss function to get accuracy of 84% on training data set.

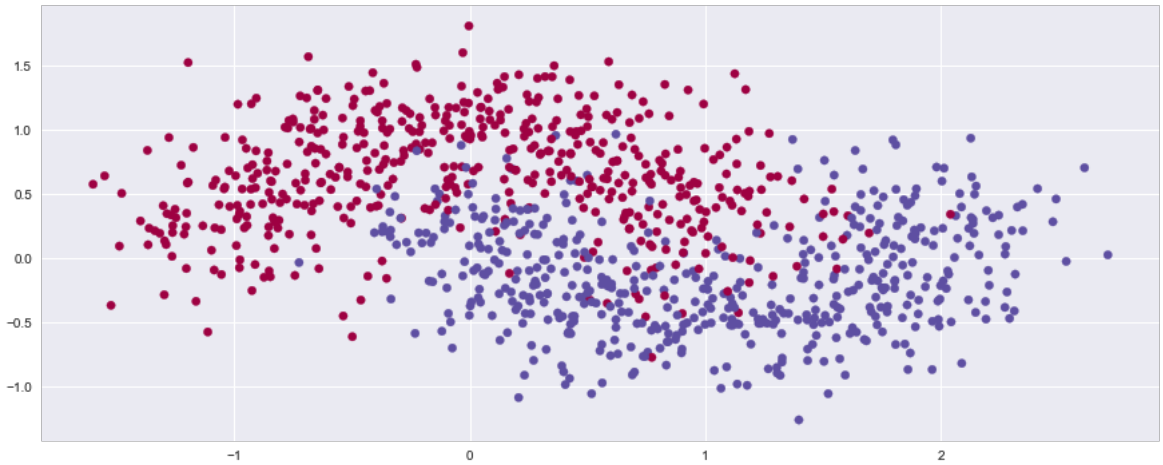


Figure 5.3: Synthetic data for purpose of explaining MAB pruning algorithms.

The following subsections describe the different MAB based pruning algorithms and use this example to illustrate the different algorithms,

5.1.1. Direct Method

The direct method [111] works by removing weights one after the other and tests the pruned network on all data sets. Based on the change of the loss function, a reward is given to each weight on the current example. Table 5-1 shows the bounded rewards using Equation 5.1 and Table 5-2 shows the binary rewards, when the weights w_{ji} are removed one by one and forward propagation used over first ten examples (from 1000 examples) of the data set.

[12] https://github.com/SalemAmeen/synthesis-Dataset/blob/master/toy%20example%20direct%20method-Second%20example.ipynb

	Example1	Example2	Example3	Example4	Example5	Example6	Example7	Example8	Example9	Example10
w_{11}	0	0	0	0	0	0	0	0	0	0
w_{12}	0	0	0	0	0	0	0	0	0	0
w_{13}	0	0	0	0	0	0	0	0	0	0
w_{14}	0	0	0	0	0	0	0	0	0	0
w_{15}	0.082	0	0.059	0	0	0	0	0	0	0
w_{16}	0	0	0	0	0	0	0	0	0	0
w_{17}	0	0	0	0	0	0.142	0	0	0	0
w_{18}	0.073	0.043	0.078	0.004	0	0	0.028	0.033	0	0
w_{21}	0	0	0	0	0	0	0	0	0	0
w_{22}	0.005	0	0	0	0	0	0	0	0	0
w_{23}	0	0	0	0	0	0.819	0	0	0.004	0.047
w_{24}	8.94E-08	0	0	0	0.019	0	0	0	0	0
w_{25}	0.071	0.006	0.061	0	0	0	0	0.001	0	0
w_{26}	0	0	0	0	0	0	0	0	0	0
w_{27}	0	0	0	0	0	0	0	0	0	0
w_{28}	0.084	0.048	0.089	0.004	0	0	0.032	0.037	0	0

Table 5-1: Cumulative average reward for bounded rewards when pruning a weight.

	Example1	Example2	Example3	Example4	Example5	Example6	Example7	Example8	Example9	Example10
w_{11}	0	0	0	0	0	0	0	0	0	0
w_{12}	0	0	0	0	0	0	0	0	0	0
w_{13}	0	0	0	0	0	0	0	0	0	0
w_{14}	0	0	0	0	0	0	0	0	0	0
w_{15}	1	0	1	0	0	0	0	0	0	0
w_{16}	0	0	0	0	0	0	0	0	0	0
w_{17}	0	0	0	0	0	1	0	0	0	0
w_{18}	1	1	1	1	0	0	1	1	0	0
w_{21}	0	0	0	0	0	0	0	0	0	0
w_{22}	1	0	0	0	0	0	0	0	0	0
w_{23}	0	0	0	0	0	1	0	0	1	1
w_{24}	1	0	0	0	1	0	0	0	0	0
w_{25}	1	1	1	0	0	0	0	1	0	0
w_{26}	0	0	0	0	0	0	0	0	0	0
w_{27}	0	0	0	0	0	0	0	0	0	0
w_{28}	1	1	1	1	0	0	1	1	0	0

Table 5-2: Cumulative average reward for binary rewards when pruning a weight.

For example, w_{15} gets 0.082 while in Table 5-2, it gets one when it is tested on the first example since the performance of the model improves when this weight is removed. In the second column, when the second example is tried, the performance does not improve and the algorithm gets zero reward. The next step restores w_{15} and tests the following weights one by one. Finally, the weights that have the most cumulative average rewards are removed.

5.1.2. Epsilon-Greedy Algorithm for Pruning the Weights

The epsilon-greedy algorithm begins by randomly selecting a weight to remove and then computing the performance on each example of the data followed by the cumulative average reward using Equation 4.3. The weight is then restored, and in the following trails, the algorithm will either choose the next weight randomly or the best weight to date depending on the value of a random number (i.e. explore). Figure 5.4 shows the MAB function (step 3) in Algorithm 5.1 where there are *K* weights, μ_{ji} denotes the current average reward, *Rnd* is a random number between zero and one.

Algorithm 5.2 Epsilon- Greedy for K arms

$Function\ \ MAB(\mu)$

Required: Parameter epsilon (0,1)

Select $a_{ji} = \begin{cases} arg\ max[\mu_{11}, \mu_{12}, \dots \mu_{ji}, \dots, \mu_{JI}] , & epsilon \leq Rnd \\ select\ an\ weight\ randomly\ from\ \{\forall K\ weights\}, & otherwise \end{cases}$

return j,i

end Function

Figure 5.4: Function of Epsilon-Greedy algorithm to prune *K* weights.

At the end of playing time, there will be a matrix of cumulative average rewards according to the performance of the model. The final step involves pruning the weights according to the cumulative average rewards matrix by setting remove those weights that have most cumulative average rewards.

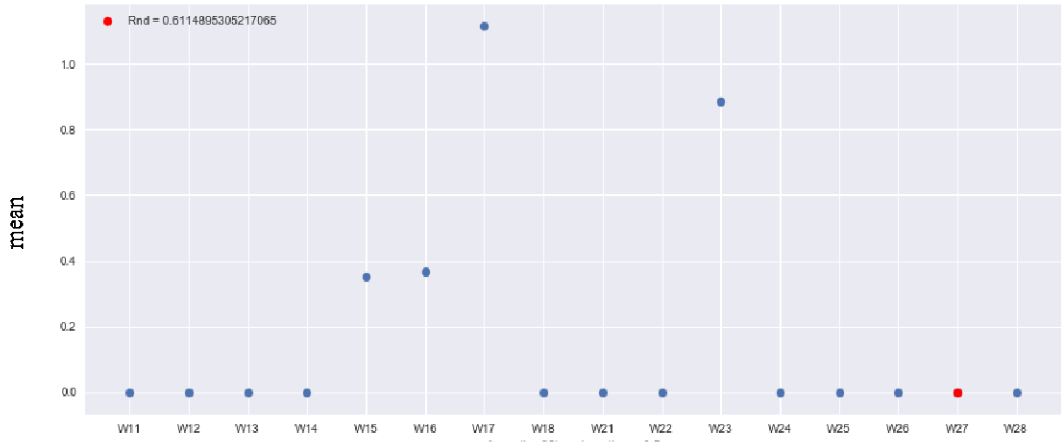
As an example, consider the example in Table 5-1 with epsilon sets to 0.5. The algorithm first explores and selects a weight randomly. Assume w_{22} is chosen and when tested on Example1, it results in a reward of 0.005.

The algorithm will update the cumulative average reward and given this weight was successful, it is kept for exploiting. The algorithm will generate a random number (Rnd) between zero and one (0.95). The generated number is greater than epsilon (0.95>0.5) so the

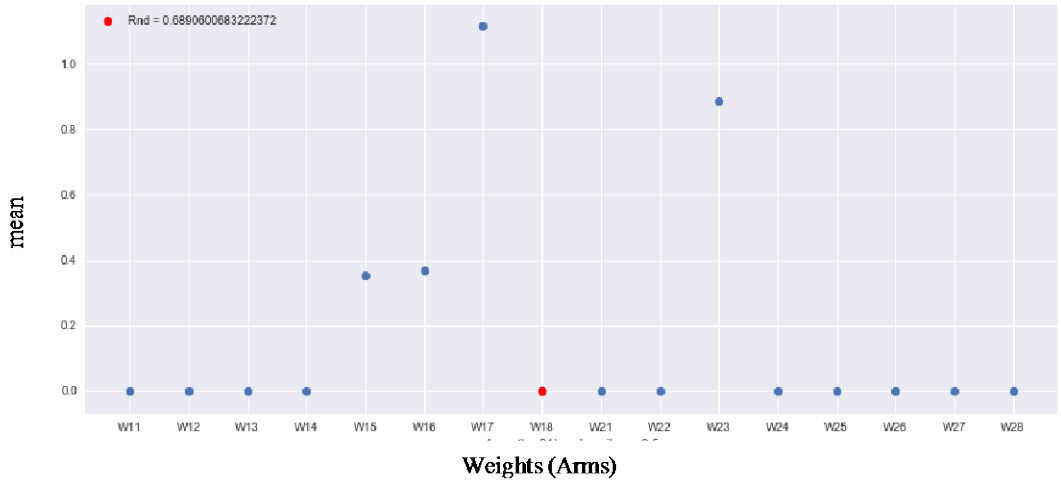
algorithm will explore and suppose that weight w_{18} is chosen randomly and forward propagation on a random example (Example3 is chosen randomly) results in a reward of 0.078. This is added to the cumulative average reward for w_{18} and its play time incremented. The algorithm continues to explore and exploit the weights until the playing time is finished. Figure 5.5 illustrates the Epsilon-Greedy algorithm while exploring and exploiting based on the Rnd values.

In the first step (Figure 5.5 (i)), Epsilon greedy begins with selecting w_{27}which results in a reward of 0. In the second step, it selects w_{18}. Steps 1 and 2 illustrates a situation where a weight is chosen randomly and step 3 shows a situation where exploitation takes place and it selects w_{17} which results in a reward of 1.18.

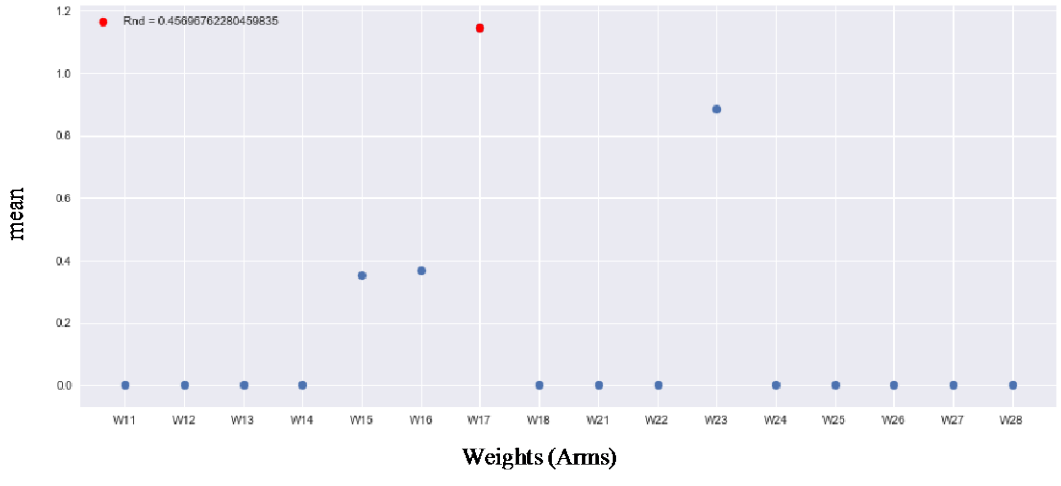
Finally, the algorithm prunes the weights that have most cumulative average rewards.



(i) t=80, epsilon=0.5 and Rnd=0.61



(ii) t=81, epsilon=0.5 and Rnd=0.69



(iii) t=82, epsilon=0.5 and Rnd=0.45

Figure 5.5: Epsilon-Greedy for pruning 16 weights at different play times. The red dots denote the chosen weight to playing. The top one played first and the bottom one played the last.

5.1.3. Win-Stay, Lose-Shift Algorithm for Pruning the Weights

The WSLS algorithm selects the weight that has highest probability to win until it loses then it explores another weight. If the data set is less noisy and the deep neural networks trained well then, we expect the less important weights will be less important over all examples or batches. However, data is often noisy in practice and the previous assumption might not work but WSLS achieves good results in machine learning and decision making applications [178, 191]. Figure 5.6 presents the WSLS function based on pursuit method [192] which is called from Step 3 of Algorithm 5.1.

Algorithm 5.3 WSLS based on pursuit method for K arms

Function $MAB(t, \mu)$
Required: Parameter $\beta \in (0,1)$
Select w_{ji} based on the highest probability P_t
Update the probabilities for all arms

$$P_{t+1}(a_{ji}) = \begin{cases} P_t(a_{ji}) + \beta(1 - P_t(a_{ji})), & if\ a\ is\ winning \\ P_t(a_{ji}) - \beta P_t(a_{ji}), & otherwise \end{cases}$$

return j,i
end Function

Figure 5.6: Function of WSLS based on pursuit algorithm to prune K weights.

To give more intuition about the algorithm we will use the example given in Table 5-1. The algorithm starts with uniform probabilities assigned to each weight, $P_i = 1/K = 0.063$. At each turn *t*, the probabilities are re-computed based Equation 4.4 and is shown in Table 5-3. The algorithm will start by pruning a weight based on the highest probability, for example w_{17} is chosen. Then, the algorithm will check the performance (on say, Example 2) after pruning w_{17}. According to Table 5-1; this will not result in a reward. The algorithm will therefore shift and explore another arm with highest probability, w_{23}, remove it and compute the change in loss and get the reward after performing forward propagation on a random example (Example3). As the reward is zero the algorithm will choose another arm with the highest probability, say w_{17} in Example1 where the reward is 0. The process will continue until the end of playing time.

	t=1 to 16 Example6			t=17 Example 2		t=17 Example 3		t=17 Example 1	
		X	P	X	P	X	P	X	P
w_{11}	0.062	0	0.050		0.050		0.050		0.050
w_{12}	0.062	0	0.050		0.050		0.050		0.050
w_{13}	0.062	0	0.050		0.050		0.050		0.050
w_{14}	0.062	0	0.050		0.050		0.050		0.050
w_{15}	0.062	0	0.050		0.050		0.050		0.050
w_{16}	0.062	0	0.050		0.050		0.050		0.050
w_{17}	0.062	0.142	0.112	0	0.089		0.089		0.089
w_{18}	0.062	0	0.050		0.050		0.050		0.050
w_{21}	0.062	0	0.050		0.050		0.050		0.050
w_{22}	0.062	0	0.050		0.050		0.050		0.050
w_{23}	0.062	0.819	0.112		0.112	0	0.089	0	0.071
w_{24}	0.062	0	0.050		0.050		0.050		0.050
w_{25}	0.062	0	0.050		0.050		0.050		0.050
w_{26}	0.062	0	0.050		0.050		0.050		0.050
w_{27}	0.062	0	0.050		0.050		0.050		0.050
w_{28}	0.062	0	0.050		0.050		0.050		0.050

Table 5-3: WSLS updated the probability P giving the reward X at each trail. The green cell represents the weight with the highest probability which will be played at the next.

5.1.4. UCB1 Algorithm for Pruning the Weights

As described in Chapter 4, initially, the UCB family the algorithm will explore pruning all the weights on the data at least once. Then, the UCB1 policy selects the weights to consider based on the upper confidence bounds. Figure 5.7 presents the function that is called from Step 3 of Algorithm 5.1.

Algorithm 5.4 UCB1 for K arms

$Function\ \ MAB(t, \mu, n, K)$

Select arm $a_{ji} = arg \max_{ij \in \{\forall K\}} (\mu_{ji} + \sqrt{\frac{2logt}{n_{ji}}})$

return j,i

end Function

Figure 5.7: Function of UCB1 algorithm to prune *K* weights.

As an illustration, consider the example given in Table 5-1. First the algorithm will play each weight at least once and receive reward. Table 5-4 shows how UCB1 proceeds where it shows each trail except the first four columns which represents the trail from 1 to 16. In each trail, the payoff X, number of playing each weight n, the cumulative average reward (μ), the padding (computed from $\sqrt{\frac{2\log t}{n_{ji}}}$) and, the sum of padding and μ which is represented by w_{ji}.

Figure 5.8 illustrates the algorithm visually, with the lines showing the range of the rewards for each weight and red lines highlighting the weights selected at each time step. For example, at *t*=79 and 80 the algorithm chooses the same weight (w_{17}) to be pruned and *t*=81, w_{27} is chosen. After the play time is finished, the weights with most cumulative average rewards removed.

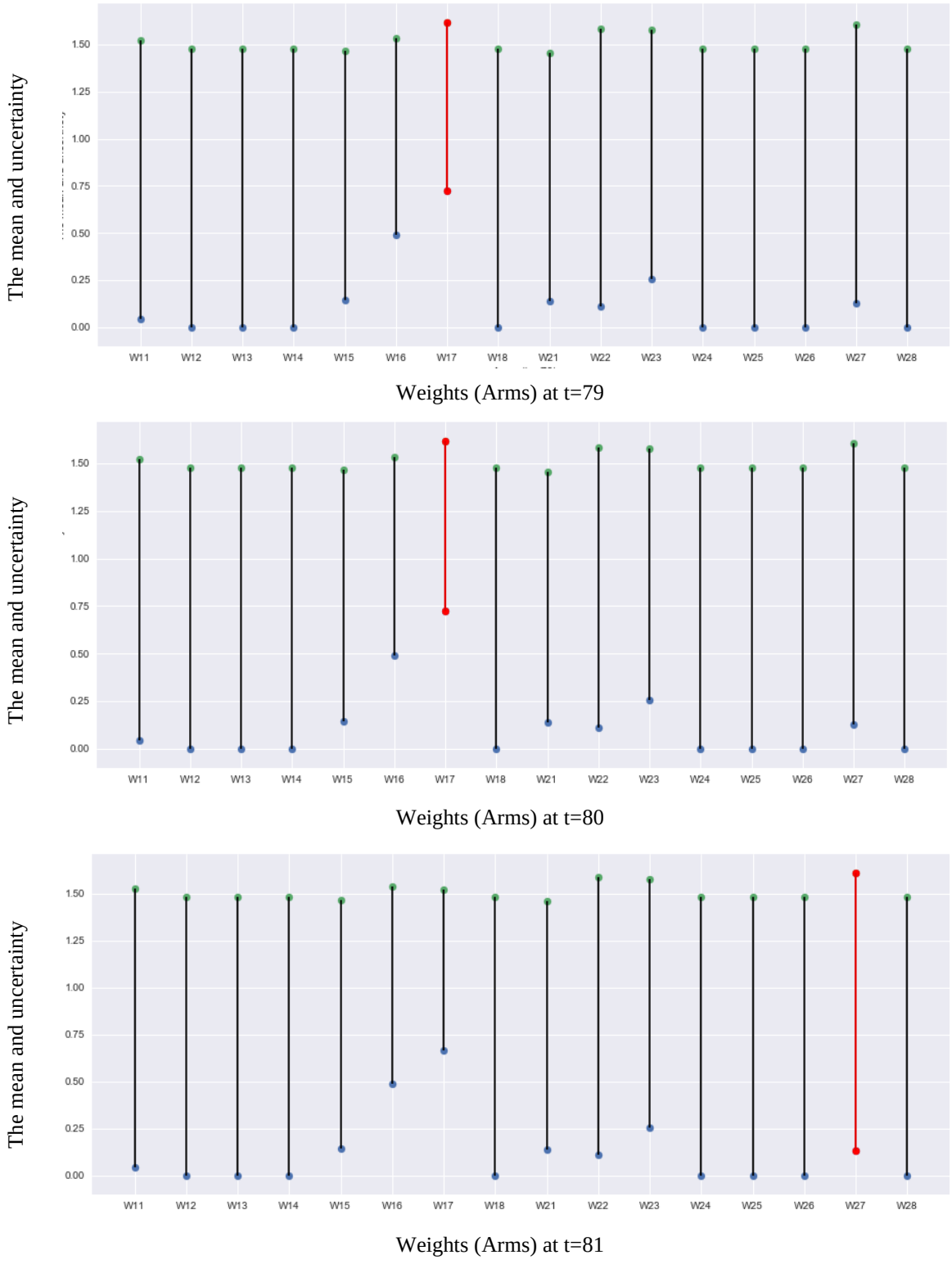


Figure 5.8: UCB1 for pruning 16 weights at different play times. Starting from the upper left till the bottom at different time. The vertical lines represent the cumulative average reward (bottom) and the padding function (top). The red line is chosen for playing.

Weight	t=1 to 16/Exampl1					t=17/Example3					t=18/Example4				
	X	n	μ_{ji}	P_q	W_{ji}	X	n	μ_{ji}	P_f	W_{ji}	X	n	μ_{ji}	P_f	W_{ji}
w_{11}	0	1	0	1.861	1.861	0	1	0	1.893	1.893	0	1	0	1.893	1.893
w_{12}	0	1	0	1.861	1.861	0	1	0	1.893	1.893	0	1	0	1.893	1.893
w_{13}	0	1	0	1.861	1.861	0	1	0	1.893	1.893	0	1	0	1.893	1.893
w_{14}	0	1	0	1.861	1.861	0	1	0	1.893	1.893	0	1	0	1.893	1.893
w_{15}	0.082	1	0.082	1.861	1.943	0.059	2	0.070	1.700	1.770	0	2	0.070	1.700	1.771
w_{16}	0	1	0	1.861	1.861	0	1	0	1.893	1.893	0	1	0	1.893	1.893
w_{17}	0	1	0	1.861	1.861	0	1	0	1.893	1.893	0	1	0	1.893	1.893
w_{18}	0.073	1	0.073	1.861	1.935	0	1	0.073	1.893	1.966	0.004	2	0.985	1.700	2.686
w_{21}	0	1	0	1.861	1.861	0	1	0	1.893	1.893	0	1	0	1.893	1.893
w_{22}	0.005	1	0.005	1.861	1.867	0	1	0.005	1.893	1.898	0	1	0.005	1.893	1.899
w_{23}	0	1	0	1.861	1.861	0	1	0	1.893	1.893	0	1	0	1.893	1.893
w_{24}	8.94E-08	1	8.94E-08	1.861	1.861	0	1	8.94E-08	1.893	1.893	0	1	8.94E-08	1.893	1.893
w_{25}	0.071	1	0.071	1.861	1.933	0	1	0.071	1.893	1.964	0	1	0.071	1.893	1.965
w_{26}	0	1	0	1.861	1.861	0	1	0	1.893	1.893	0	1	0	1.893	1.893
w_{27}	0	1	0	1.861	1.861	0	1	0	1.893	1.893	0	1	0	1.893	1.893
w_{28}	0.084	1	0.084		0.084	0	1	0.084	0	0.084	0	1	0.084	1.893	1.978
Max					1.943					1.966			0.985		2.686

Table 5-4: UCB1 method where X is the reward, n number of plays, t is the total playing time so far, P_f is the padding function and w_{ji} is the weights (the algorithm will choose the value). μ_{ji} is cumulative average reward and green colour cell is the arm will be played next.

5.1.5. KL-UCB Algorithm for Pruning the Weights

In KL-UCB, the padding is computed by going beyond the mean, $\mu(a)$ of an arm a by a certain distance based on the Kullback-Leibler measure (d). That, is KL-UCB seeks the maximum q that satisfies the following:

$$d(\mu_{ji}, q) \leq \frac{logt + c\ log\ logt}{n_{ji}}$$

Where n_{ji} is the number of times weight has been pulled and *t* is the number of rounds and *c* is constant (recommended to be zero [180, 181]).

For example, for weight w_{11} in Figure 5.10, KL-UCB seeks the largest q that meets:

$d(0.26, q) \leq \frac{64}{4} = 1.04$.

Figure 5.9 shows the function this is called from Algorithm 5.1.

Algorithm 5.5 KL-UCB for K arms

Function $MAB(t, \mu, n, K)$
Select $a_{ij} = arg \max_{ji \in \{\forall K\}} (d(\mu_{ji}, q) \leq \frac{logt + c\ log\ logt}{n_{ji}})$
return j,i
end Function

Figure 5.9: Function of KL-UCB algorithm for pruning the *K* weights.

Figure 5.10 represents determining the value of q based on $d(\mu_{ji}, q) \leq \frac{logt + c\ log\ logt}{n_{ji}}$. The horizontal (orange) line represents the value of $\frac{log(t)}{n_{ji}}$ and the curve (blue) line represents the $d(\mu_{ji}, q)$. For example, the weight w_{12} has tighter ($\frac{log(t)}{n_{12}}$) at trail 66 than at trails 64 and 65.

All the weights are considered and the one with the maximum q is selected.

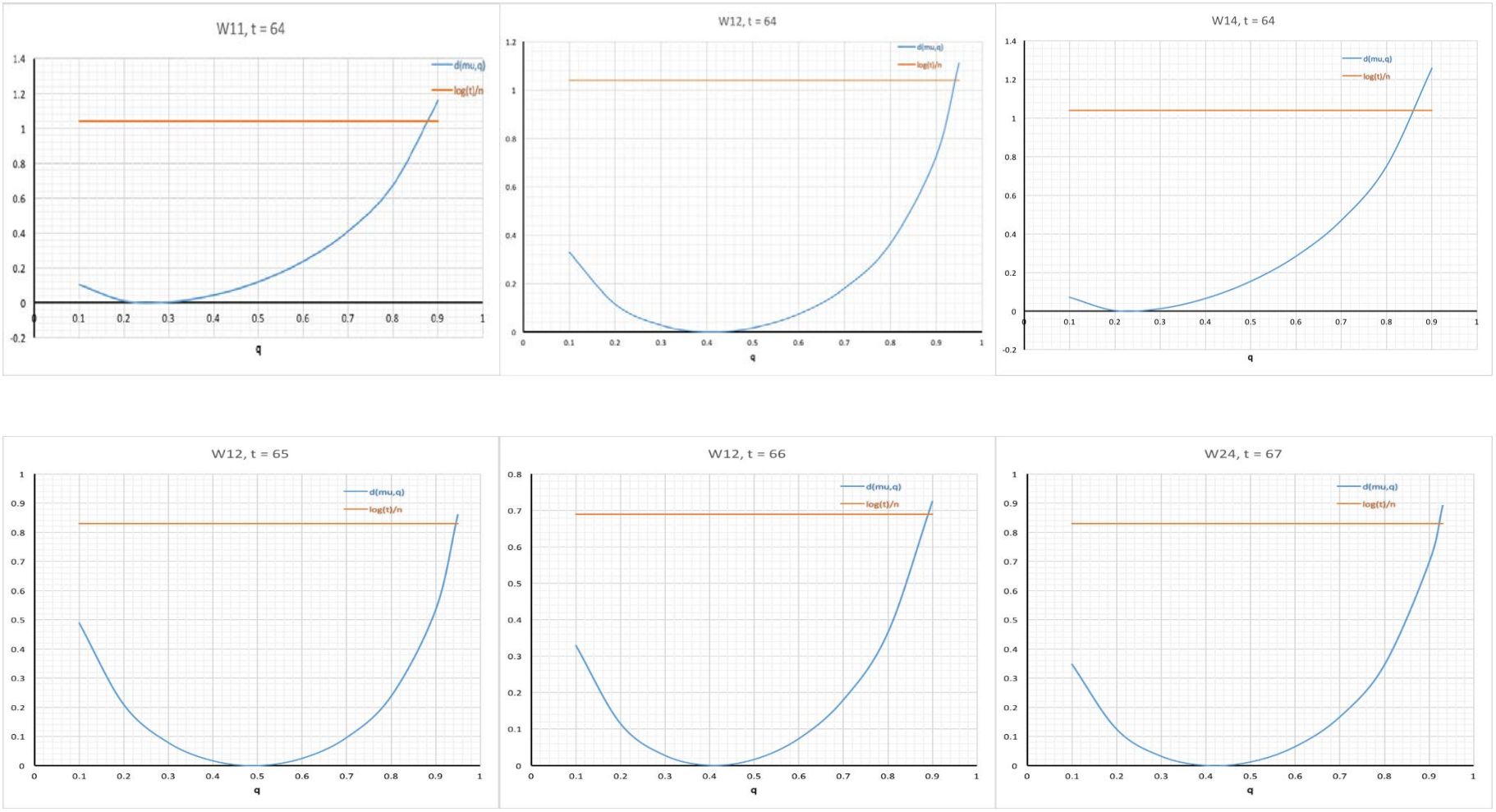


Figure 5.10: compute q of the weight where the charts on the top represent the weights at the play time between (t=49 to t=64). Then, the charts at the bottom represent computing the maximum q for the current chosen weight.

5.1.6. Thompson Sampling Algorithm for Pruning the Weights

Thompson Sampling [193] selects the next weight based on drawing a random sample from a distribution representing the prior knowledge of the weights. This is then used to assign a reward after evaluating the loss function.

Given S_a, the number of times the arm a yields a positive reward and F_a, the number of times an arm fails to yield a reward the probability of succeeding is drawn from a Beta distribution, $Beta(x;\ S_a + 1, F_a + 1)$ where

$$Beta(x; \alpha, \beta) = \frac{\Gamma(\alpha + \beta)}{\Gamma(\alpha)\Gamma(\beta)} x^{\alpha-1}(1-x)^{\beta-1}$$

While Γ is called Gamma function and the mean and variance are given by

$$\mu = \frac{\alpha}{\alpha + \beta}$$

$$\sigma^2 = \frac{\alpha\beta}{(\alpha + \beta)^2(\alpha + \beta + 1)}$$

Figure 5.11 presents the function that is called from the algorithm in Algorithm 5.1 based on Thompson Sampling with binary rewards for *K* arms. Giving the earlier example, the algorithm first plays every weight once. Then, the success S_{ji} and the failure F_{ji} are updated and the random sample is drawn from the Beta distribution of each weight.

Table 5-5 illustrates the algorithm. For example, at time *t*=16, the weight that has the highest number is played the next which is weight w_{22}.

Algorithm 5.6 Thompson Sampling for binary bandits for K arms

```
Function MAB(t, S, F, K)
        Repeat for all weights (∀K weights)
            Sample θ_ji(t) for Beta(S_ji + 1, F_ji + 1) distribution
        end for
        Select a_ij = arg max_{ji∈{∀K}} θ_ji(t)
        return j,i
end Function
```

Figure 5.11: Thompson Sampling where there are *K* weights and w_{ij} is the weight selected to play next.

At next trail *t*=17, weight w_{28} is chosen as it has the largest sample which is drawn from Beta distribution based on total success and failure. At *t*=18, the weight w_{28} is chosen for

playing. Then, at *t*=19, w_{23} is chosen and so on. Figure 5.13 illustrates the kind of distributions that can arise and examples of samples that are drawn.

5.1.7. BayesUCB Algorithm for Pruning the Weights

Bayesian UCB (BayesUCB) is based on maintaining, updating and using a probability distribution for each arm, where the probability distribution takes the form of the beta distribution defined by the number of successes and failures of each arm S_a^{t-1}, F_a^{t-1}, where *t* is the number of rounds. The upper bound used for each weight is determined by using the quantile function:

$$Q(\alpha,\lambda) \text{ such that } P(X \leq Q(\alpha,\lambda)) = \alpha$$

Where λ is the distribution and α is the level. BayesUCB uses increasingly tighter bounds as the number of rounds increases with α set to $1 - \frac{1}{t}$.

The algorithm prunes the weights depending on the rewards gained during the playing time and returns the current weight for playing. Then, the algorithm returns the weight that has the largest quantile. Figure 5.12 illustrates the function that replaced MAB function in 5.2 based on BayesUCB.

Algorithm 5.7 BayesUCB for K arms

$Function\ \ MAB(t, S, F, K)$
Required: λ_{ji}^{t-1} which is define by S and F
Repeat for all weights ($\forall K\ weights$)
Compute $q_{ji}(t) = Q(1 - \frac{1}{t}, \lambda_{ji}^{t-1})$
end for
Select $a_{ji} = arg \max_{ji \in \{\forall K\}} q_{ji}(t)$
return j,i
end Function

Figure 5.12: Function of BayesUCB to prune *K* weights.

Table 5-6 shows the results of applying BayesUCB on the example in Table 5-2 at different play times. At time *t*=16, the algorithm will choose w_{15} as it has the largest quantile. Then, at *t*=17,18 and 19 the weights w_{18}, w_{28} and w_{22} are chosen Figure 5.14 illustrates how the quantile function is used to select between different weights.

Weight	t=1to16/Example1				t=17/Example3				t=18/Example10				t=19/Example8			
	X	S	F	Sample (beta)	X	S	F	Sample (beta)	X	S	F	Sample (beta)	X	S	F	Sample (beta)
w_{11}	0	0	1	0.335	0	0	1	0.156	0	0	1	0.546	0	0	1	0.299
w_{12}	0	0	1	0.057	0	0	1	0.158	0	0	1	0.390	0	0	1	0.215
w_{13}	0	0	1	0.011	0	0	1	0.250	0	0	1	0.011	0	0	1	0.551
w_{14}	0	0	1	0.500	0	0	1	0.559	0	0	1	0.068	0	0	1	0.119
w_{15}	1	1	0	0.621	0	1	0	0.831	0	1	0	0.912	0	1	0	0.446
w_{16}	0	0	1	0.035	0	0	1	0.374	0	0	1	0.149	0	0	1	0.112
w_{17}	0	0	1	0.535	0	0	1	0.368	0	0	1	0.681	0	0	1	0.020
w_{18}	1	1	0	0.559	0	1	0	0.867	0	1	0	0.506	0	1	0	0.156
w_{21}	0	0	1	0.527	0	0	1	0.555	0	0	1	0.238	0	0	1	0.566
w_{22}	1	1	0	0.888	0	1	1	0.190	0	1	1	0.424	0	1	1	0.090
w_{23}	0	0	1	0.152	0	0	1	0.130	0	0	1	0.352	0	0	1	0.869
w_{24}	1	1	0	0.886	0	1	0	0.849	0	1	1	0.748	0	1	1	0.567
w_{25}	1	1	0	0.767	0	1	0	0.474	0	1	0	0.703	0	1	0	0.070
w_{26}	0	0	1	0.056	0	0	1	0.270	0	0	1	0.870	0	0	1	0.286
w_{27}	0	0	1	0.000	0	0	1	0.149	0	0	1	0.132	0	0	1	0.097
w_{28}	1	1	0	0.601	0	1	0	0.910	0	1	1	0.953	1	2	1	0.589
Max				0.888				0.910				0.953				0.869

Table 5-5: Results of Thompson Sampling where X is the current binary reward for each weight, t the total play time, S is the success, and F is failure and Sample (beta) is drawn from the beta distribution for each weight. At each time step, the algorithm will choose the weight that has the highest reward among the others which is shown in the green cell.

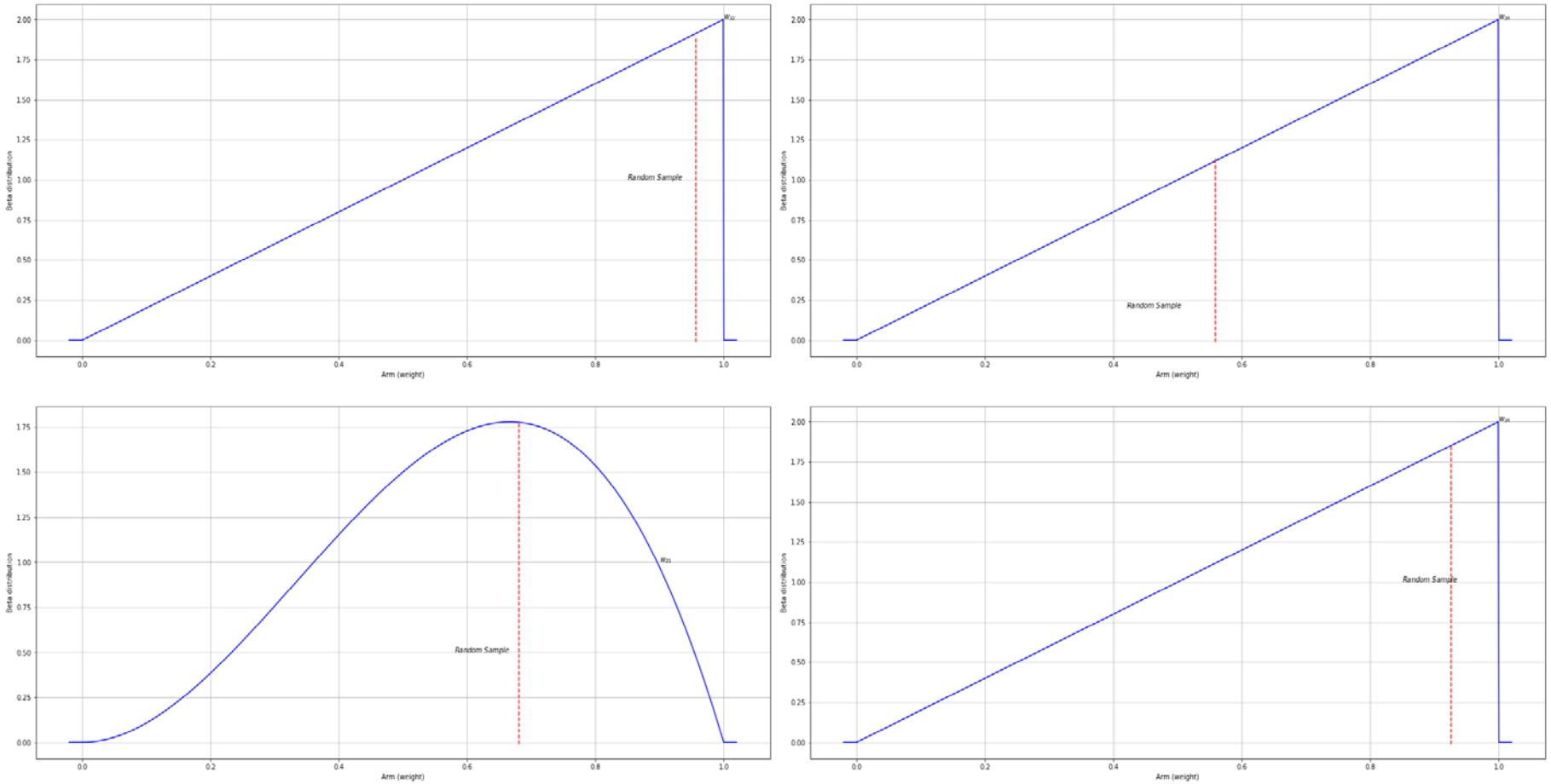


Figure 5.13: Thompson Sampling for choosing the arm to play next based on the sample from beta distribution. The two arms on the top are chosen from the first column in the previous table while the charts in the bottom are chosen from the last column of the same table. On the top, the algorithm will choose the arm on the left as it has higher reward while on the bottom the algorithm will choose the arm on the right as it has higher reward.

Weight	t=1to16/Example1					t=17/Example5					t=18/Example6					t=19/Example10				
	X	S	F	1-(1/t)	Quantile (beta)	X	S	F	1-(1/t)	Quantile (beta)	X	S	F	1-(1/t)	Quantile (beta)	X	S	F	1-(1/t)	Quantile (beta)
w_{11}	0	0	1	0.9375	0.750	0	0	1	0.941	0.757	0	0	1	0.944	0.764	0	0	1	0.947	0.771
w_{12}	0	0	1	0.9375	0.750	0	0	1	0.941	0.757	0	0	1	0.944	0.764	0	0	1	0.947	0.771
w_{13}	0	0	1	0.9375	0.750	0	0	1	0.941	0.757	0	0	1	0.944	0.764	0	0	1	0.947	0.771
w_{14}	0	0	1	0.9375	0.750	0	0	1	0.941	0.757	0	0	1	0.944	0.764	0	0	1	0.947	0.771
w_{15}	1	1	0	0.9375	0.968	0	1	1	0.941	0.853	0	1	1	0.944	0.857	0	1	1	0.947	0.861
w_{16}	0	0	1	0.9375	0.750	0	0	1	0.941	0.757	0	0	1	0.944	0.764	0	0	1	0.947	0.771
w_{17}	0	0	1	0.9375	0.750	0	0	1	0.941	0.757	0	0	1	0.944	0.764	0	0	1	0.947	0.771
w_{18}	1	1	0	0.9375	0.968	0	1	0	0.941	0.970	0	1	1	0.944	0.857	0	1	1	0.947	0.861
w_{21}	0	0	1	0.9375	0.750	0	0	1	0.941	0.757	0	0	1	0.944	0.764	0	0	1	0.947	0.771
w_{22}	1	1	0	0.9375	0.968	0	1	0	0.941	0.970	0	1	0	0.944	0.972	0	1	0	0.947	0.973
w_{23}	0	0	1	0.9375	0.750	0	0	1	0.941	0.757	0	0	1	0.944	0.764	0	0	1	0.947	0.771
w_{24}	1	1	0	0.9375	0.968	0	1	0	0.941	0.970	0	1	0	0.944	0.972	0	1	0	0.947	0.973
w_{25}	1	1	0	0.9375	0.968	0	1	0	0.941	0.970	0	1	0	0.944	0.972	0	1	0	0.947	0.973
w_{26}	0	0	1	0.9375	0.750	0	0	1	0.941	0.757	0	0	1	0.944	0.764	0	0	1	0.947	0.771
w_{27}	0	0	1	0.9375	0.750	0	0	1	0.941	0.757	0	0	1	0.944	0.764	0	0	1	0.947	0.771
w_{28}	1	1	0	0.9375	0.968	0	1	0	0.941	0.970	0	1	0	0.944	0.972	0	1	1	0.947	0.861
Max					0.750					0.970					0.972					0.973

Table 5-6: Results of BayesUCB on different play time where X is the current binary reward for each weight, t the total play time, S is the success, and F is failure and Quantile is drawn from the beta distribution for each weight with probability 1-(1/t). At each time step, the algorithm will choose the weight that has the highest quantile among the others which is shown in the green cell.

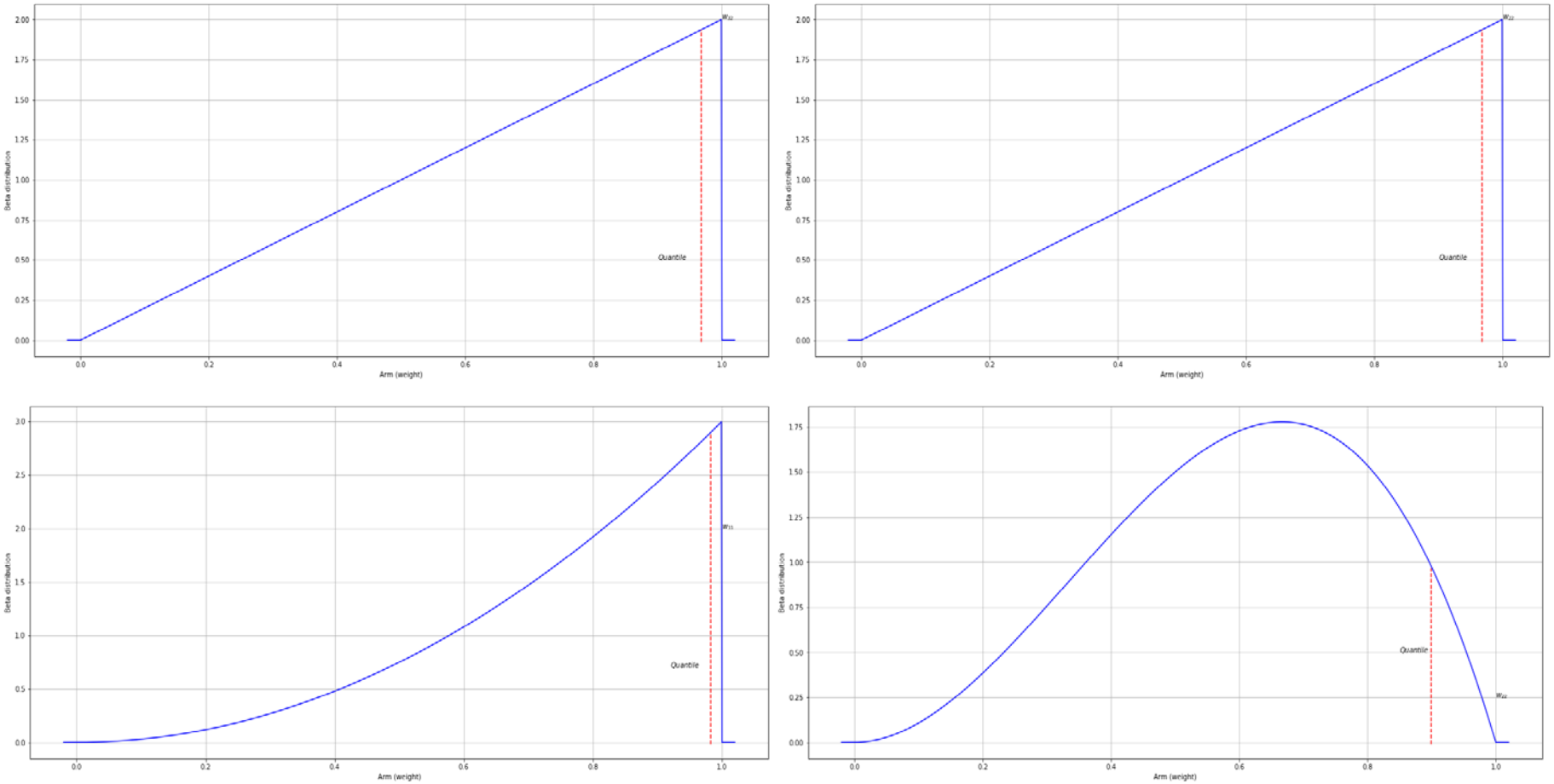


Figure 5.14: BayesUCB for choosing the arm to play next based on the sample from the beta distribution. The two arms on the top are chosen from the first column in Table 5-5 while the charts at the bottom are chosen from the last column of the same table. On the top, the algorithm will choose the arm on the left as it has higher quintile while on the bottom the algorithm will choose the arm on the right as it has higher quintile

5.2. Evaluation

We train our models using stochastic gradient decent with a batch size of 10, momentum of 0.9, and weight decay is set to 0.005. The learning rate is initialized as 0.01. The Softmax function is used as the activation function. The experiments were implemented on 12-core Intel(R) Core i7-5820 3.30-GHz and 64 GB RAM. The methods developed and implemented using TensorFlow and Python.

The experimental methodology involved carrying out two sets of experiments. In the first, the NNSYSID[13] package was used to build neural networks for 12 data sets from the UCI machine learning repository as shown in Table 5-7. The inputs and outputs reflected the features and classes of the data sets. For consistency, each network adopted two hidden layers, with each hidden layer utilizing 20 neurons. The neural networks were then pruned using the different methods and their performance analysed. In the second experiment, LeNet's deep learning model [83] with two convolutional layers and two fully connected layers was adopted and trained on the MNIST data set. The model was then pruned using the different methods and the methods compared.

The following subsections present the results from the two sets of experiments. The methodology for the comparison is based on the recommendations by Demšar [194] who advocate the use of a non-parametric test due to Friedman test [195] to determine if there is a difference amongst the methods, and if so, to follow up with the use of the Nemenyi test [196] to assess if one method is significantly better than another.

[13] http://www.iau.dtu.dk/research/control/nnsysid.html

Full name of the Data set	No. of examples	No. of features	Number of classes
Banknote Authentication	1,372	4	2
Blood Tra. Service Centre	748	4	2
Credit Approval	690	15	2
Haberman's Survival	306	3	2
Liver Disorders	345	6	2
MAGIC Gamma Tele.	19,020	10	2
Mammographic Mass	961	5	2
MONK's Problems	432	6	2
Connectionist Bench	208	59	2
Spambase	4,601	56	2
SPECTF Heart	267	44	2
Tic-Tac-Toe Endgame	958	9	2

Table 5-7: UCI data sets

5.2.1. Results from the Experiments on the UCI Data sets

The empirical evaluation on the UCI data is carried out in comparison with the recent and most famous algorithms:

i. Random Pruning, which is the simplest algorithm, and involves random selection of weights that are pruned, an evaluation of the pruned network and then a decision on whether the removal of the weight had a positive or negative effect.

ii. A Network Pruning method [134] that removes weights that are below a user specified threshold value.

iii. Optimal Brain Damage (OBD), a method developed by LeCun et al. [125], that was one of the first methods for reducing the size of neural networks. OBD removes the weights that if set to zero would have least effect on the training error. To measure the effect of changing weights, LeCun et al. [125] used a Taylor series approximation for the change in error that would occur if the weights were perturbed (as described in Chapter 3). This analysis leads to the need to solve a Hessian matrix which can be computationally expensive. To reduce this computational cost, LeCun et al. [125] ignored the off-diagonal values.

iv. Optimal Brain Surgeon (OBS), a method due to Hassibi et al. [126-128] in which the Hessian matrix that results from the analysis for OBD is solved without making the assumption that the off-diagonal elements can be ignored.

Table 5-8 presents the errors $\in$ (where the accuracy is 100-$\in$) for each of the pruning methods on data sets from the UCI repository. after pruning 20% of the weights using the different methods.

Data set	Model	E. Greedy	WSLS	UCB1	KL-UCB	TS	BayesUCB	OBD	OBS	Pruning Network	Random
Banknote Authentication	0.86	0.86	0.89	0.86	0.86	0.86	0.86	0.89	0.87	4.08	5.98
Blood Tra. Service Centre	0.94	0.93	1.05	0.93	0.93	0.93	0.93	0.93	0.93	1.29	0.93
Credit Approval	0.97	0.95	0.93	0.93	0.96	1.63	0.96	1.98	9.47	3.4	23.04
Haberman's Survival	0.94	0.93	0.94	0.93	0.93	0.93	0.93	0.93	0.93	1.48	1.5
Liver Disorders	0.95	0.95	0.96	0.95	0.95	0.95	0.95	0.95	1.7	1.47	1
MAGIC Gamma Tele.	0.91	0.95	1.17	0.91	0.91	0.91	0.91	0.92	0.97	3.34	1.28
Mammographic Mass	0.94	0.94	0.94	0.94	0.94	0.94	0.94	0.94	0.94	3.44	0.98
MONK's Problems	0.95	0.97	1.14	0.95	0.95	0.95	0.95	0.95	6.13	1	0.98
Connectionist Bench	0.97	1.14	0.97	0.97	0.97	1.92	0.97	0.97	0.97	1.01	1.01
Spambase	0.93	0.95	0.94	0.94	0.94	0.94	0.94	0.95	5.22	2.52	5.86
SPECTF Heart	0.91	0.92	0.91	0.91	0.91	0.93	0.91	0.91	0.99	13.1	0.91
Tic-Tac-Toe Endgame	0.91	0.91	2.43	0.91	0.91	0.91	0.91	0.91	0.92	22.15	12.77

Table 5-8: Computed error on validation data set before and after pruned the model. The green cell shows the method with less error while red cell shows the method with large error. The arrows point up if the error high, down if it is low or in right direction if it is in between.

Table 5-9 presents the average rank of the 11 methods over the 12 data sets, with UCB1 being ranked the most effective in terms of reducing the error rate and the Network Pruning method the least effective. Applying the Friedman test results in a p-value of 6.99×10^{-10} confirming there is a significant difference amongst at least some of the methods.

Name of Method	Mean Rank
UCB1	3.75
Bayesian UCB (BayesUCB)	4.00
KL-UCB	4.00
Model before pruning (Model)	4.62
Thompson Sampling (TS)	5.20
OBD	5.25
Epsilon-Greedy (E. Greedy)	5.62
Win-Stay, Loss-Shift (WSLS)	6.79
OBS	7.70
Random Pruning (random)	9.00
Network Pruning	10.04

Table 5-9: Results of the average rank of the methods on 12 different data sets.

Figure 5.15 depicts the results from the Nemnyi post-hoc test. The methods are plotted according to their average rank, where the best ranked methods are to the left with the Critical Difference (CD=4.33). For example, UCB1 has mean plus CD equal to nearly 8 which is statistical better than Network Pruning with mean 10.

The CD for the Nemenyi test is calculated from:

$$q_{\alpha,K}\sqrt{\frac{K(K+1)}{6N}}$$

Where α is the confidence which is set to 0.05, K is the number of models (or classifiers), and N the number of measurements (data sets). To compute $q_{\alpha,K}$, the Studentized range statistic for infinite degrees of freedom divided by squared root of two is used[14].

The horizontal lines group the methods that are not significantly different when the Nemnyi post-hoc test is used at the 0.05 level. These results show that:

- The UCB family of methods performed significantly better than Random Pruning and Network Pruning method.

[14] http://kourentzes.com/forecasting/wp-content/uploads/2014/05/nemenyi.csv

- Use of Thompson Sampling for pruning also performed significantly better than Network Pruning method.
- The performance of BayesUCB and KL-UCB is very similar, which is consistent with the theoretical results due to Kaufmann et al. [197].
- Although the bandit based methods have a higher average rank, the Nemnyi test does not distinguish these methods significantly from OBD or OBS in terms of minimizing the error.

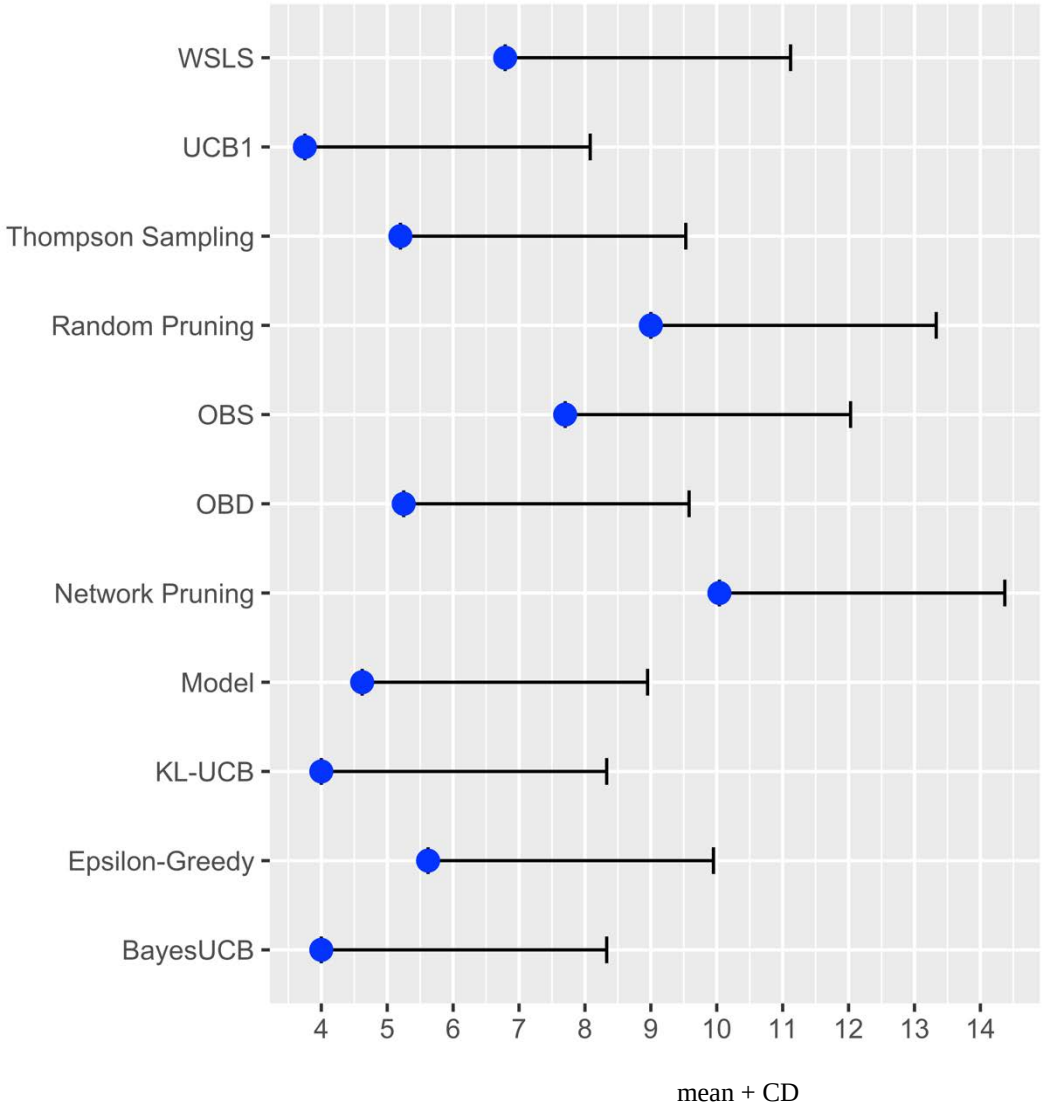


Figure 5.15: Comparison of all classifiers against each other with the Nemenyi test. Lines show the critical difference for each method any Groups of classifiers that are not significantly different (at p = 0.05) are out of the lines. The blue dot shows the rank mean while the line determine the CD which 4.33.

Table 5-10 presents the run-time performance of these methods, showing that the UCB family, WSLS, Epsilon-Greedy and Thompson Sampling have the best run-time performance which is followed by OBD. OBD and OBS are more computationally intensive given the need to compute Hessian matrix. Thus, given the ranking of the methods given in Figure 5.15, UCB methods achieve better performance on average than OBD and OBS but in significantly less time.

Per Second	OBS	OBD	EG	WSLS	UCB1	TS	BayesUCB	KL-UCB
Banknote Authentication	15.13	14.26	2.04	1.08	0.81	1.05	4.61	4.02
Blood Tra. Service Centre	14.72	13.94	1.41	0.88	0.76	1.06	3.98	3.34
Credit Approval	235.63	235.51	3.89	4.11	3.91	4.84	15.09	11.5
Haberman's Survival	12.06	11.08	1.38	0.79	0.61	0.94	3.44	2.9
Liver Disorders	23.16	22.08	1.14	0.74	0.63	0.99	4.65	3.79
MAGIC Gamma Tele.	51.57	50.97	2.48	1.58	0.79	9.89	10.02	9.69
Mammographic Mass	15.1	13.96	1.49	0.92	0.77	1.09	4.1	3.31
MONK's Problems	28.51	27.73	1.03	0.86	0.76	1	4.99	4.15
Connectionist Bench	983.66	921.76	0.89	1.19	0.95	2.88	24.63	13.46
Spambase	252.05	249.87	3.83	4.04	3.97	4.81	15.81	12.24
SPECTF Heart	512.14	492.53	0.89	1.14	0.95	2.41	19.61	10.78
Tic-Tac-Toe Endgame	36.99	36.23	1	1	0.87	1.29	5.36	4.87
Average	181.73	174.16	1.79	1.53	1.32	2.69	9.69	7
Standard Deviation	282.67	278.31	1.08	1.21	1.23	2.69	7.28	4.12

Table 5-10: Run-time performance in seconds for the different pruning methods on different data sets. Green cell shows the methods that have less computation time while the red cell shows the ones with the highest computation time.

5.2.2. Results for the MNIST Data set

The MNIST (Modified National Institute of Standards and Technology) data set is a well-known collection of handwritten digits [83] that has been used in evaluating many

handwriting recognition algorithms [15]. One of the most widely adopted deep learning architectures for this data set and problem is the LeNet model [83]. In this model, the network has two 5x5 convolutional layers with 20 and 50 filters respectively, and two fully connected layers with 500 and 10 (output layer) neurons. Table 5-11 summarizes the number of parameters including the weights and biases in each layer of this model. The activation and pooling layers have no parameters.

Layer		Parameters	Weights
Layer1	Convolution (Conv)	520	500
Layer2	Convolution (Conv)	25,050	25,000
Layer3	Fully connected (FC)	2,500,500	2,500,000
Layer4	Fully connected (FC)	5,010	5,000
Total		2,531,080	2,530,500

Table 5-11: No of parameters in the LeNet's model.

The base line accuracy of this model is 98.06%, so it provides a good example for assessing which methods can best remove weights without adversely affecting the level of accuracy. To assess this, we apply a selection of methods to prune 50% of the weights in the layers 2 and 3 which have the most weights. The methods we select include: one from the UCB1 given their performance is similar, Thompson Sampling, Random Pruning, and the Network Pruning Method. The OBD and OBS methods are not selected given that they are not applicable, as they are needed to compute the Hessian matrix and invers Hessian matrix respectively. In addition, the RELU activation function is used to train LeNet and the second derivative of it is zero.

Table 5-12 presents the results, showing the use of the bandit algorithms maintains accuracy although the use of Network Pruning and Random Pruning does result in a significant decline in accuracy. The first column of the table has the model and five different pruning methods. The second column has the first fully connected layer (layer3) in LeNet model while the last column holds the second convolution layer (layer2) in the LeNet model.

[15] http://yann.lecun.com/exdb/mnist/

	FC	Conv
Model	0.9806	0.9806
TS Prune 50%	0.9830	0.9810
EG Prune 50%	0.9808	0.9807
UCB1 Prune 50%	0.9840	0.9820
Random Pruning 50%	0.5330	0.4120
Network Pruning 50%	0.5920	0.4950

Table 5-12: Results of pruning 50% of two layers in the LeNet's Model.

5.3. Summary

This chapter presented the six proposed methods for pruning the weights. These methods are based on the idea of using MAB for optimizing between exploration and exploitation. The experimental analysis and evaluation of the MAB pruning algorithms presented in this chapter, was conducted using the UCI data sets and the MNIST data. The main findings from the evaluation of the proposed methods indicated that:

- In general, the MAB pruning algorithm produced better results than the original models. In Table 5-9 UCB1, BayesUCB and KL-UCB achieve better ranking than the original model and in MNIST data set, there is slightly improve on accuracy over the original unpruned model even though half of the model parameters is pruned.
- Some of the proposed methods outperformed the other pruning methods. That is, UCB based methods performed best and WSLS based methods performed worst.
- The proposed methods had manageable time to prune the weights of the network in contrast to other methods like OBS and OBD.

The next chapter extends the methods developed in this chapter so they can be used to prune neurons.

6. Multi-Armed Bandits for Pruning Neurons

When the number of parameters are very large, removing a weight at a time is time-consuming. Instead, removing a neuron with all its weights one at a time could be more effective. Hence, in this chapter, different MAB algorithms are tested and compared to state of the art pruning techniques on many different data sets. The chapter is organized as the follows. Section 6.1 presents the top-level algorithm form pruning neurons, Section 6.2 presents pruning the neurons based on the change in loss function. Section 6.3 presents an evaluation of the pruning algorithms on different data sets and Section 6.4 presents a summary of this chapter.

6.1. Summary of MAB Algorithm for Pruning Neurons

Figure 6.1 presents the top-level algorithm for pruning neurons. The basic idea for pruning neurons is the same as that for pruning weights, except that a MAB algorithm is used to select and remove neurons instead of weights.
The initial step of the algorithm, selects the layer to be pruned. Even though MAB pruning algorithms can be used to prune all neurons together, we will use these algorithms to prune the neurons in specific layers given the time and computation limitations, especially of deep neural networks.

The following summarises the main steps of this algorithm:

1 and 2, iterate over all playing time and determine the random example respectively.

3 determines a neuron that needs to be removed according to a MAB.

4 and 5, computes the performance of the model based on a random example of the data. That is, it computes the loss function based on an example of the data $L(D|N)$ where N is a non-zero neurons in the network and D is a random example from the training data set. Hold the value of neuron.

6, 7 and 8, removes the determined neuron (from step 3) and tests the performance. Computes $L(D|N')$ where N' is the network without the neuron selected in step 3. Then, restores the neuron for the next play.

9 and 10, computes the change in the loss function δL. If there is improvement then there is a positive reward (+1 if binary rewards), otherwise there will be no reward (zero). Then, the algorithm will update the cumulative average rewards for the playing arm (removed neuron).

11 update the cumulative average reward.

12 and 13, compute the weights if the called function is EXP3 or Hedge function.

14 checks the playing time; if it has not ended, the process is repeated; otherwise, stops playing.

15 finally, the last step removes unwanted neurons that have the most cumulative average rewards, which in fact have least effect on the performance.

As described in Chapter 4, there are many MAB algorithms and a number of these were tried for removing weights and the results presented in Chapter 5. The lessons learned from the evaluations presented in Chapter 5 were used to select the MAB algorithms to evaluate:

- Epsilon-Greedy algorithm was chosen primarily because of its simplicity although it also showed good performance.
- UCB1 and Thompson Sampling were chosen given that both resulted in good accuracy and good run time performance.

- KL-UCB and BayesUCB were not selected given they require significant computational time.
- WSLS had the worst performance among MAB pruning algorithms so was not selected.
- EXP3, Hedge and Softmax algorithm are used in this chapter. EXP3 and Hedge are based on adversarial bandits [170, 175, 187, 188] where the reward is not stochastic. When a neuron is pruned, the expected reward might change over the time with giving same input.

The following subsections summarise the direct method, and the new three MAB based pruning algorithms (softmax, Hedge and EXP3). In each case, an example is given to illustrate the methods for pruning neurons. The examples use the synthetic data set given in Figure 5.3.

Algorithm MAB 6.1 Algorithm for pruning the neurons

INPUT: Time horizon T, Trained network, Input layer l to be pruned.
OUTPUT: Pruned network
Let X_i be the current reward of this arm/neuron
Let w_i is the weight for each arm (neuron) this is not the neural networks' weights but it is the parameter for Adversarial bandit algorithms and initialized to one, In addition, Parameter $\epsilon \in [0,1]$, Parameter $\gamma \in [0,1]$
Let μ_i be the cumulative average reward of this arm/neuron, initialized to zero
Let D the random example from training data set
Let N_i be the neuron i in layer l
Let $L(D|N)$ The loss function before pruning the neuron
Let $L(D|N')$ The loss function after pruning the neuron
Let Threshold how much loss in the performance is allowed
Let K is the number of neurons in the chosen layer
Let n_i is the total current play time for the N_i
Let S_i is the success and F_i is the failure, both initialized to zero

1. for t=1 to T do /* start playing */
2. D = random example of the training data
   if any $n_i = 0$ then choose the current index i /* Pull each neuron at least once */
3. else Call the relevant MAB algorithm, returning the index of the selected neuron:
   $i = MAB(*arg)$
4. Perform forward propagation on D to compute $L(D|N)$
5. Hold the value of the selected neuron, $temp = N_i$
6. Set the neuron to zero $N_i = 0$
7. Perform forward propagation on D to compute $L(D|N')$
8. Set the value of the neuron to previous value, $N_i = temp$
9. $\delta L = L(D|N) - L(D|N')$
10. $X_{i,t} = REWARD(\delta L)$
11. Update the cumulative average reward of the current arm
    $\mu_{i,t+1} = (n_i - 1)/n_i * \mu_{i,t} + 1/n_i * X_{i,t}$
12. if MAB is Hedge algorithm then $\rho_i = X_{i,t}$, $w_{i,t+1} = w_{i,t}(1+\epsilon)^{\rho_i}$
13. if MAB is EXP3 algorithm then $\rho_i = X_{i,t}$, $w_{i,t+1} = w_{i,t}.e^{\gamma.\frac{\rho_i}{p_i.K}}$
14. end for
15. $PrunedModel = PrunedFunction(model, \mu)$
16. end main program

17. $Function\ PrunedFunction(model, rewards)$

18. Set to zero the neurons that have most rewards
19. return PrunedModel
20. end Function

21. $Function\ REWARD(\delta L)$

22. if (bounded reward) then /* For example, reward for UCB1*/
23. $Reward += max(0, \delta L + Threshold)\ /Constant$
24. else /* Reward for Thompson Sampling */
25. if $\delta L < 0$ then reward=0, $F_{i,t+1} = F_{i,t} + 1$
26. else reward=1, $S_{i+t+1} = S_{i,t} + 1$
27. end if
28. end if
29. end Function

Figure 6.1: The generic algorithm of a MAB pruning neurons.

6.1.1. Direct Method

The direct method also known as brute-force pruning [114] works by removing neurons one after the other and then testing the pruned network on all the data set. Table 6-1 shows the bounded rewards using Equation 5.1 when the neurons *N* are removed one by one and forward propagation is used over the first three examples (from 1000 examples) of the data set.

Neuron	Example1	Example2	Example3	Average Reward over three examples
N_1	1.92	2.65	2.51	2.36
N_2	0	1.2	0	0.4
N_3	0	0	0	0
N_4	0.91	2.01	1.74	1.55

Table 6-1: Cumulative average reward for the bounded rewards when pruning a neuron on example of data at each forward propagation.

Once the rewards for the selected neurons have been computed, the neuron with the highest average reward is removed. In the case of considering only theses three examples, N_1 is pruned.

6.1.2. Softmax Algorithm for Pruning the Neurons

The general steps of pruning the neurons using Softmax algorithm [174] are the same as that is explained in Section 6.1 except for step 3 where Softmax algorithm will be used to determine the neuron that needs to be removed as following:

1. Choose the arm based on probability P_i.
2. Update the probability for each arm using Equation 4.5.

Figure 6.2 shows the Softmax function that is called from Algorithm 6.1.

Algorithm 6.2 Softmax for K arms

$Function\ \ MAB(t, \mu, K)$
Required: Parameter $\tau \in \mathbb{R}$
Select neuron N_i with highest probability $exp^{\frac{\mu_i}{\tau}} / \sum_{k=1}^{K} exp\frac{\mu_k}{\tau}$
return i
end Function

Figure 6.2: Function of Softmax algorithm to prune *K* neurons.

To illustrate how the Softmax bandit prunes neurons, consider the example in Table 6-1. First, we assume the algorithm will check to prune every neuron once and test the change in the loss function by performing forward propagation on a random example. Table 6-2 shows the steps of pruning the neurons based on Softmax bandit.

The table presents the state after round *t*=4, showing cumulative average reward (μ_i), the number of attempts at pruning each neuron (*n*) and the probability of a reward (*P*). The next neuron chosen is then based on the one with highest probability of a reward. For example, N_1 is chosen as it has the highest reward (0.422).

Neuron	t=1 to 4 / Example1				
	X	n	μ_i	$exp^{\frac{\mu_i}{\tau}}$	P
N₁	1.92	1	1.92	2.611	0.422
N₂	0	1	0	1	0.161
N₃	0	1	0	1	0.161
N₄	0.91	1	0.91	1.576	0.254

Table 6-2: Softmax function for pruning the neurons where μ_i is cumulative average reward, X is given reward, $\tau = 2$ and P is the probability.

6.1.3. Hedge Algorithm for Pruning the Neurons

Instead of computing the probability of the rewards in Softmax function, the Hedge algorithm maintains weights for each neuron. These weights are similar to those found in algorithms

such as AdaBoost (and not the weights of a neural network). First, the weights initialized to one and then later they are computed using Equation 4.12.

The probability of reward is then computed as proportion of the weight over the sum of the weights of all the neurons.

Figure 6.3 shows the Hedge function that will replace the MAB function call in Algorithm 6.1. Notice that the weights are passed as an argument from Algorithm 6.1 and are also updated in Algorithm 6.1.

Algorithm 6.3 Hedge Algorithm K arms

Function $MAB(w, K)$

Select neuron N_i with highest probability $P_i = \frac{w_i(t)}{\sum_{j=1}^{n} w_j(t)}$

return i

end Function

Figure 6.3: The Hedge function for pruning *K* neurons.

Table 6-3 gives an example of the Hedge algorithm in operation with ϵ sets to 0.05. First, the algorithm will receive a reward ρ for the neuron that is played. Then the weights are updated and the probability of getting a reward from each neuron are calculated. Finally, the algorithm will choose the neuron based on the probability as shown in Figure 6.3, where neuron N_1 is chosen for time steps 4 to 6.

Weight	t=1to4/Sample1			t=5/Sample3			t=6/Sample2		
	ρ	w	P	ρ	w	p	ρ	w	P
N_1	1.92	1	0.25	2.51	1.13	0.27	2.65	1.28	0.30
N_2	0	1	0.25		1	0.24		1	0.23
N_3	0	1	0.25		1	0.24		1	0.23
N_4	0.91	1	0.25		1	0.24		1	0.23
sum		4			4.13			4.28	

Table 6-3: The steps of choosing next neuron to prune based of Hedge algorithm. The green cell is the probability of choosing the following neuron. ρ is the generated non-stationary reward, w is the weight and P is the probability for choosing the next neuron. $\epsilon = 0.05$.

6.1.4. EXP3 Algorithm for Pruning the Neurons

The EXP3 algorithm is similar to the above adversarial algorithms except that the calculation for the probabilities is different. That is, the weights (as used in Hedge) are updated using Equation 4.13 and the probabilities are calculated using Equation 4.14. Figure 6.4 shows the EXP3 function that will replace MAB function in the Algorithm 6.1. Notice that the weights are passed as an argument from Algorithm 6.1 and are also updated in Algorithm 6.1

Algorithm 6.4 EXP3 Algorithm K arms

$Function\ \ MAB(w, \gamma, K)$

Select neuron N_i with highest probability $p_i = (1-\gamma)\frac{w_i}{\sum_{i=0}^{K} w_j} + \gamma.\frac{1}{K}$

return i

end Function

Figure 6.4: EXP3 function to prune k neurons.

To illustrate EXP3, consider the example in Table 6-1 with γ=0.1. First, the algorithm will assign one to the weight for each neuron and then update the probability of randomly choosing the neuron to prune, as shown in fourth column in Table 6-4. Following that, the first neuron (N_1) is chosen to prune with probability 0.25. At step t=5 the algorithm will update the weight for the chosen neuron and the weights for the other neurons will be the same as in the previous play.

Weight	**t=1to4/Sample1**			**t=5/Sample3**			**t=6/Sample2**		
	ρ	w	P	ρ	w	p	ρ	w	P
N_1	1.92	1	0.25	2.51	1.285	0.294	2.65	1.609	0.339
N_2	0	1	0.25		1	0.235		1	0.220
N_3	0	1	0.25		1	0.235		1	0.220
N_4	0.91	1	0.25		1	0.235		1	0.220
sum		4			4.285			4.609	

Table 6-4: EXP3 for pruning the neurons where ρ is the current non-stationary reward, w is the weight and p is the probability for choosing the next neuron to play. γ=0.1. Green cells are the neurons chosen to prune.

6.2. Evaluation

This section presents the results of an empirical evaluation of the MAB based methods for pruning neurons. All proposed MAB algorithms were implemented using the Python programming language. All the reported experiments in this chapter were conducted using NVIDIA TITAN X. The evaluation is carried out in two stages.

First, sixteen data sets from the UCI and Kaggle data were utilised and their characteristics are summarised in in Table 6-5. The methodology for these data involved randomly splitting the data into three folds: a training, validation and test fold (60%, 20%, 20% respectively). The training fold is used for optimizing the parameters of the classifiers (models), the validation fold for hyperparameter optimization, and the test fold for evaluation. When a data set was small (less than 1000 examples), then it was randomly divided into 80% training data, 20% testing. Ten-fold Cross Validation was used on 80% training data for building and validating the models. The evaluation measures used were accuracy, f1 score, precision and recall, providing a broader insight into the effectiveness of the individual classifiers. In addition, AUC (Area Under the receiver operating Curve) and confusion matrices were also used. For conciseness, the body of the thesis focuses presents the results in terms of accuracy but the other measures, such as f1 score, precision, recall and AUC are included in Appendix1. In these data sets, forward neural networks were trained.

The second set of experiment utilised data that had been used for developing deep learning models. Eight different data sets (taken from the different resources that can be used for benchmarking deep learning algorithms) were used and their characteristics are shown in Table 6-6. Most of these data sets were already split in advance into training and testing folds. The evaluation measure used for these experiments was accuracy. These data sets are based on images or texts and as is common practice, ConvNets were used for pixel data and RNNs were used for sequence data.

After obtaining the data set, some pre-processing was performed because this is known to improve convergence when training [83]. Next, the architecture of a neural network was designed and the number of parameters used was based on a rule of thumb that suggests that

a full model should have at least the same number of parameters as there are examples in the training data set [198] for example, for the Reuters data, there are 8.982 examples so 558 neurons and weights were used 535,552.

As Chapter 2 describes, there are several methods for training neural networks. In this work, the mini-batch gradient descent method was used. Following some initial experimentation, the Adam method, as recommended by Kingma & Ba [63], was used with a learning rate of $1e^{-3}$ for optimisation for most of the data sets unless better results were obtained by other optimizers at validation. However, other methods were also tried and Table 6-7 lists the optimisation method and hyperparameters used for each data set.

Once a model was trained, it was available for the experiments to compare the MAB pruning algorithms. All the MAB based pruning algorithms developed need to decide the number of plays. In the experiments below, the play time was set to at least as much as twice the number of neurons (arms) in the played layer. Once a network was pruned, its accuracy was measured on a test set.

Data set	Full name of the Data set	No. of examples	No. of features	Number of classes
Pima	Pima Indians Diabetes	768	9	2
Car	Car Evaluation	1,728	6	4
Spambase	Spambase	4,601	56	2
Adult	Adult	48,842	14	2
Valley	Hill-Valley	606	100	2
Titanic	Titanic	1,309	11	2
Face	Labelled Faces in the Wild	1,288	1,850	7
Wine	Wine	178	12	3
Heart	Heart Disease	303	13	2
Iris	Iris	150	3	3
Abalone	Abalone	4,177	8	3
Poker	Poker Hand	1,015,010	8	3
Glass	Glass Identification	214	10	7
Wine Quality	Wine Quality	4,898	11	10
Chest	Activity Recognition from Single Chest-Mounted Accelerometer	50,000	3	7
Cancer	Lung Cancer	569	32	2

Table 6-5: Small data set specification.

Data set	Full name of the Data set	No. of features	No. of examples in training	No. of examples in testing	Number of classes
ImageNet	ImageNet	256×256×3	1,200,000	150,000	1,000
IMDB	Internet Movie Review	5,000	25,000	25,000	2
Reuters	Reuters newswire topic classification task	1,000	8,982	2,246	5
MNIST	Mixed National Institute of Standards and Technology	28×28×1	60,000	10,000	10
Cifar-10	Cifar-10	32×32×3	50,000	10,000	10
Cifar-100	Cifar-100	32×32×3	50,000	10,000	100
SVHN	Street View House Numbers	32×32×3	73,257	26,032	10
bAbI	bAbI	20,000	10,000	1,000	2

Table 6-6: Data set specification for deep learning models.

Data set	Optimizer	Batch size	Dropout	Weight decay	Neurons
Pima	Adadelta	8	None	None	35
Car	Adam	6	None	L2(0.001)	25
Spambase	RMSProp	10	0.25	L2(0.01)	40
Adult	Adadelta	100	None	None	60
Valley	Adam	100	0.5	L2(0.001)	40
Titanic	Adam	9	None	L2(0.001)	60
Face	Adam	100	0.5	None	300
Wine	Adam	13	0.5	L2(0.001)	35
Heart	Adam	13	None	None	35
Iris	Adam	1	0.5	L2(0.001)	16
Abalone	Adam	100	None	L2(0.0001)	30
Poker	Adam	20	0.5	L2(0.001)	25
Glass	Adam	100	0.5	L2(0.001)	35
Wine Quality	Adam	100	0.5	L2(0.001)	25
Chest	Adam	13	None	L2(0.001)	20
Cancer	Adam	30	0.5	L2(0.001)	30

Table 6-7: Hyperparameters of neural networks trained on different data sets. In addition, the learning rate for all of them is set to 0.001, the activation function is ReLU and the number of epochs is 100.

The data sets and methodology used are summarised above. The main objectives of the evaluation are as follows:

1. To measure the effectiveness of the proposed pruning algorithms (pruning based on Epsilon-Greedy, Softmax, UCB1, Thompson Sampling, Hedge and EXP3), we compare the proposed methods against each other including the original unpruned models.
2. To compare the classification effectiveness of the proposed methods and different classifiers, the performance of the methods was compared with techniques like SVM, Decision trees, KNN, LDA, Naïve Bayes, QDA, Gaussian process classifier and Logistic regression. For completeness, the performance of the algorithms was also compared with grouping and boosting algorithms like bagging, random forests, Adaboost, LightGBM and Xgboost.
3. To compare the effectiveness of the proposed methods relative to other recent techniques for pruning networks. This includes comparison with a greedy algorithm presented in Polyak & Wolf [119] and an algorithm that prunes neurons based on magnitude Kruschke [113] which has recently been shown to have good results [134, 146].

The results obtained are presented in the following subsections. These are organised as follows. Subsection 6.2.1 presents the results obtained using MAB pruning algorithms to prune forward neural networks developed for the UCI and Kaggle data. Subsection 6.2.2 presents the results obtained using pruning algorithms on the deep learning data.

6.2.1. Results from the Experiments on the UCI data sets

This subsection presents the results of comparing the MAB pruning algorithms with each other and the unpruned networks: The experimental methodology was described above and the detailed steps taken are documented in Appendix 1 for completeness. Some of the MAB

algorithms have user specified hyperparameters. After a few trials[16], these were set as follows:

- The UCB1 and Thompson Sampling algorithms do not need hyperparameters except the number of plays.
- We use the formula $\frac{1}{\log(t+\varphi)}$ [176] where $\varphi = 0.00001$ to decay epsilon in the Epsilon-Greedy algorithm and also use it to decay the temperature in Softmax.
- Some MAB algorithms have constant hyperparameter like Epsilon-Greedy where we set epsilon to 0.9, EXP3 where we set gamma to 0.2 and in Softmax, we set the temperature to 0.9. These hyperparameter were selected after many experiments which suggested that these settings worked well. Generally we find these are worked good with many data sets. addition, the number of experiments are large so we will let using other values of these parameters by annealing method.

Figure 6.5 shows a comparison between several MAB pruning algorithms on two data sets as the number of neurons pruned increases (Appendix 3 and 4 includes all the other results). In Figure 6.5, the behaviour of the algorithms is mostly the same on the different data sets. In general, UCB1 shows the most stable algorithm among the others on most of the data sets. At the beginning of pruning, all proposed algorithms outperform the original unpruned model then the performance decreases over the time. On face data set, SVM, decision tree, KNN and neural networks are only algorithms were being used as this data set suffer from curse of dimensionality [199] compared to the other data set.

Table 6-8 shows the results of the accuracy on the different proposed methods and other classifiers. In addition, Appendix 3 presents further details about accuracy, f1 score, precision and recall results.

In general, the results show that pruning neuron networks with MAB methods can improve performance (accuracy, f1 score, precision and recall) and can work better than some other classifiers trained on the same training data set.

[16] Experiment different hyperparameters on Face and Iris data sets.

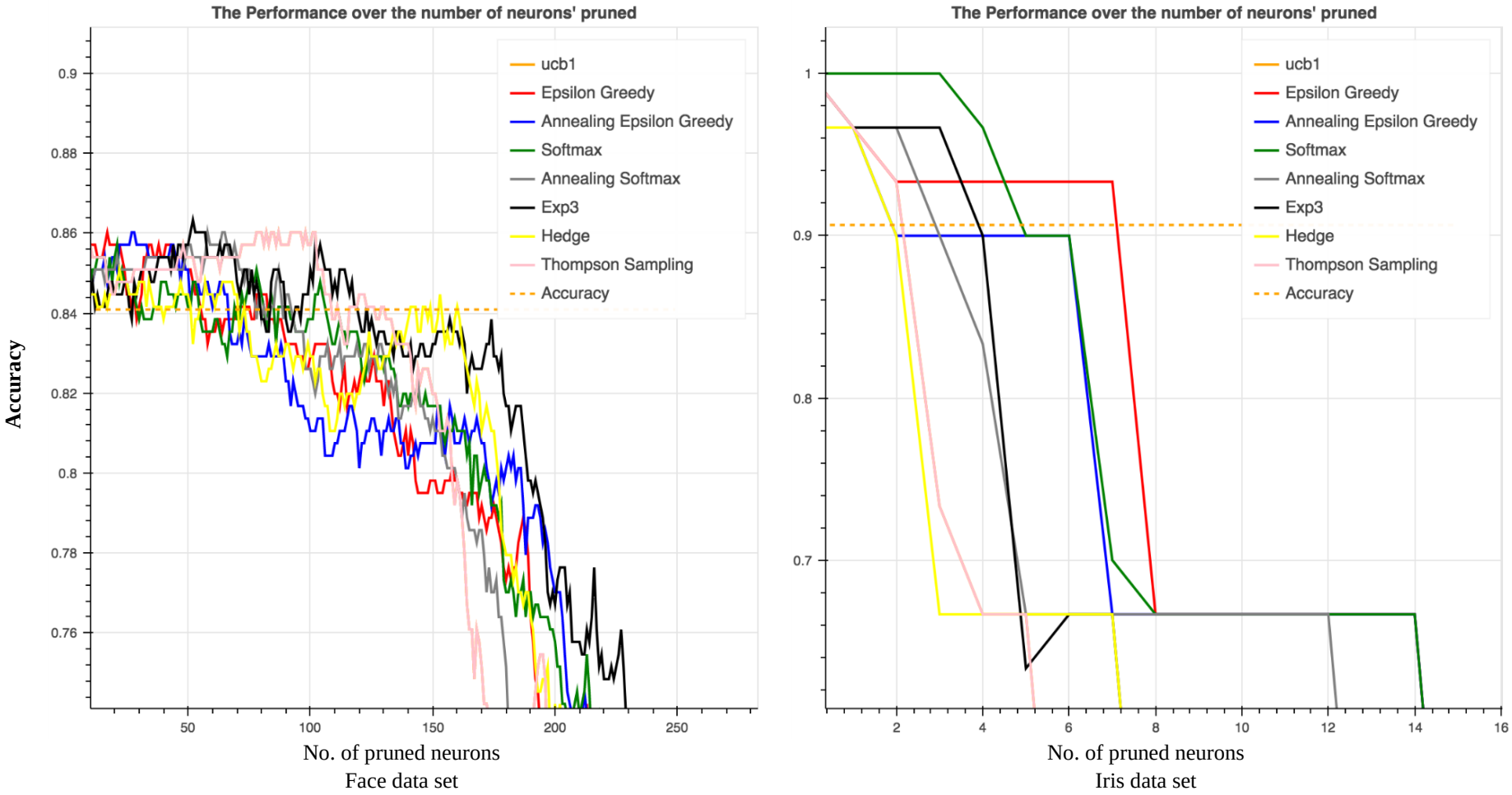


Figure 6.5: Results of MAB pruning Algorithms.

Methods	SPAM	ABALONE	ADULT	CANCER	CAR	GLASS	HEART	IRIS	PIMA	POKER	TITANIC	VALLY	WINE	WINE QUALITY	FACE	CHEST
Knn	0.909	0.541	0.817	0.93	0.954	0.628	0.542	1	0.688	0.47	0.732	0.624	0.972	0.538	0.696	0.575
LSVM	0.921	0.538	0.798	0.947	0.847	0.558	0.508	0.933	0.63	0.274	0.777	0.573	1	0.529		0.32
SVM	0.945	0.579	0.834	0.974	0.951	0.581	0.75	1	0.721	0.454	0.788	0.589	1	0.575	0.835	0.54
DT_gini	0.894	0.476	0.788	0.947	0.931	0.465	0.725	1	0.643	0.62	0.67	0.569	0.889	0.532	0.388	0.505
DT_entorpy	0.911	0.469	0.797	0.93	0.951	0.512	0.767	0.9	0.688	0.614	0.637	0.607	0.889	0.552		0.55
Bagging Knn	0.933	0.557	0.822	0.965	0.786	0.628	0.575	0.967	0.643	0.446	0.754	0.601	0.972	0.578		0.44
Bagging DT	0.915	0.545	0.812	0.956	0.789	0.581	0.733	0.867	0.649	0.448	0.771	0.581	0.889	0.562		0.415
Random Forest	0.89	0.511	0.829	0.965	0.899	0.512	0.775	1	0.708	0.674	0.76	0.601	0.889	0.602		0.625
Ada Boost	0.746	0.541	0.817	0.939	0.887	0.558	0.483	0.933	0.669	0.288	0.777	0.581	0.889	0.415		0.375
NB	0.807	0.514	0.787	0.807	0.832	0.535	0.492	0.933	0.597	0.272	0.771	0.548	0.917	0.471		0.36
LDA	0.916	0.562	0.785	0.965	0.838	0.581	0.508	0.967	0.649	0.272	0.771	0.538	0.944	0.54		0.33
QDA	0.813	0.555	0.802	0.912	0.905	0.512	0.567	0.967	0.63	0.34	0.804	0.564	1	0.49		0.365
Log. Reg.	0.911	0.565	0.797	0.956	0.847	0.535	0.517	0.967	0.649	0.272	0.777	0.573	1	0.534		0.345
GP Class.	0.926	0.559	0.837	0.947	0.928	0.605	0.8	1	0.695	0.674	0.771	0.609	0.917	0.603		0.53
LightGBM	0.925	0.538	0.831	0.965	0.902	0.512	0.583	0.933	0.721	0.378	0.782	0.612	0.917	0.547		0.585
Xgboost	0.845	0.569	0.837	0.947	0.928	0.535	0.725	0.933	0.669	0.337	0.788	0.614	0.917	0.554		0.59
NN	0.945	0.58	0.835	0.965	0.954	0.605	0.608	0.967	0.688	0.342	0.799	0.583	1	0.552	0.854	0.455
UCB1	0.948	0.573	0.832	0.965	0.936	0.651	0.642	1	0.701	0.361	0.799	0.587	1	0.558	0.854	0.405

Table 6-8: Comparison of accuracy between pruning based on UCB1 and different classifiers. The results with respect to some classifiers are not available in the case of the Face Data set because of the resource required. The green cells indicate that the method has good accuracy in contrast of red cell. The arrows point up if the error high, down if it is low or in right direction if it is in between.

We utilised all the major statistical comparison measures to confirm the performance of the results. These include the accuracy, f1 score, precision and recall measures. Eight proposed methods were tested against the unpruned model and the other classifiers. The p value obtained from applying the Friedman test on the accuracy, f1 score, precision and recall results was 4.49×10^{-15}, 2.07×10^{-08}, 3.98×10^{-08} and 1.25×10^{-06} respectively. All p values are less than 0.05, indicating that there is significant difference between the methods. Table 6-9 shows the results in rank order of performance according to accuracy.

Name of Method	Mean Rank			
	Accuracy	F1 score	Precision	Recall
SVM	20.218	18.844	20.156	18.656
UCB1	19.063	18.625	16.500	18.063
Trained Model (NN)	18.531	17.843	15.593	17.219
Gaussian Process (GP Class.)	16.906	17.344	16.781	17.219
Tomp. Sampling	16.313	16.031	13.781	15.531
Decay ε Greedy (Decay E Gr.)	15.719	14.531	15.094	14.313
ε Greedy (E Greedy)	15.625	15.063	14.188	14.625
Softmax	15.156	13.969	14.625	13.719
Decay Softmax (Decay SM)	14.969	14.531	16.625	12.719
Random Forest	14.500	14.031	16.594	12.750
Hedge	14.406	13.531	13.938	13.031
Knn	14.156	14.438	13.125	14.312
LightGBM	14.094	12.250	12.063	10.906
Xgboost	13.813	12.469	15.188	11.938
Bagging Knn	13.188	11.500	13.000	11.031
EXP3	12.281	12.344	12.094	11.563
Decision Tree (DT_entorpy)	11.156	13.563	10.906	14.469
Bagging DT	10.407	9.719	11.906	8.125
Decision Tree Gini (DT_gini)	9.875	13.656	10.781	16.063
QDA	8.844	12.250	9.281	13.031
Logistic Regression (Log. Reg.)	8.781	7.594	10.781	8.406
LDA	8.313	8.656	9.813	8.313
Linear SVM (LSVM)	7.594	7.438	8.594	8.875
Ada Boost	6.907	8.031	8.094	10.406
Naïve Bayes (NB)	4.188	7.594	5.406	9.719

Table 6-9: Results of ranked accuracy, f1 score, precision and recall results based on Nemenyi test, which is used to compare the different models on 16 different data sets. In this table, the highest is the better and the table is sorted on the accuracy column.

In general, Table 6-9 demonstrates that SVM outperforms the other classifiers and that neural networks are second-best compared to other classifiers. Pruned neural networks,

based on UCB1 outperform the original unpruned models on these data sets. Other pruning methods like Thompson Sampling, Epsilon-Greedy and Softmax pruned the original model with some loss in the performance. The pruning algorithms based on adversarial bandits, like EXP3, perform poorly.

Given the performance of the algorithms is different, the next step is to apply the Nemenyi test between all the methods. These results are shown in Figure 6.6, Figure 6.7, Figure 6.8 and Figure 6.9 for the accuracy, f1 score, precision and recall respectively. In Figure 6.6 and Figure 6.7, it is clear that pruning the networks made the performance in terms of both the accuracy and f1 score statistically better than bagging decision trees. In terms of precision, Figure 6.8, shows that pruning using UCB1 outperformed Adaboost. Finally, in terms of the recall, Figure 6.9 shows that pruning statistically improved the network to become better than Naïve Bayes.

The results for these tests and the implementation are available online[17].

[17] https://github.com/SalemAmeen/testing_python_friedman/blob/master/prune_neurons/classification/friedman.ipynb

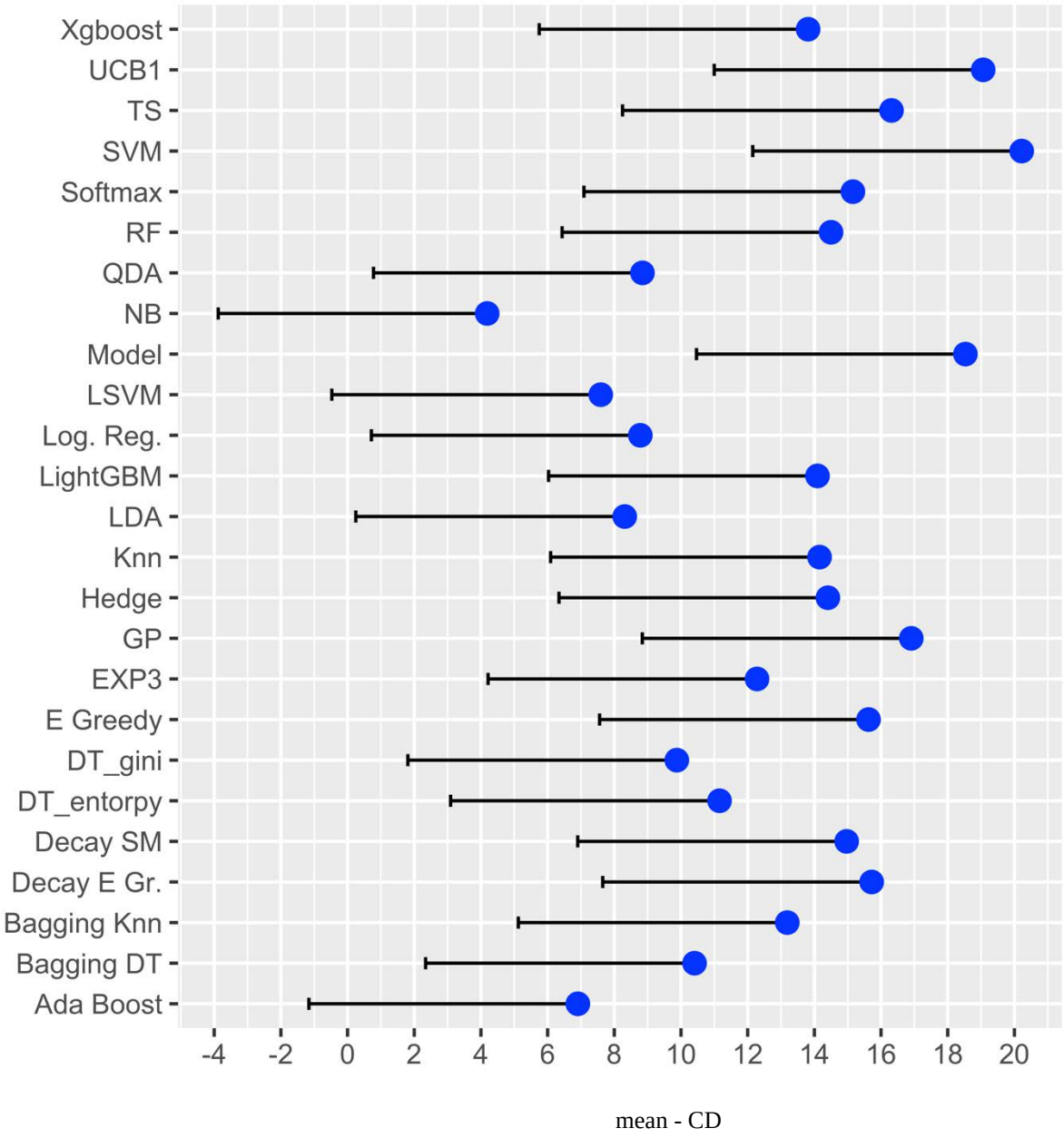


Figure 6.6: Comparison of the accuracy of all classifiers against each other with the Nemenyi test. Horizontal lines show the critical difference away from proposed pruning methods and any other methods. Groups of classifiers that are not significantly different (p = 0.0.5) are out of the lines from proposed methods. CD=8.066.

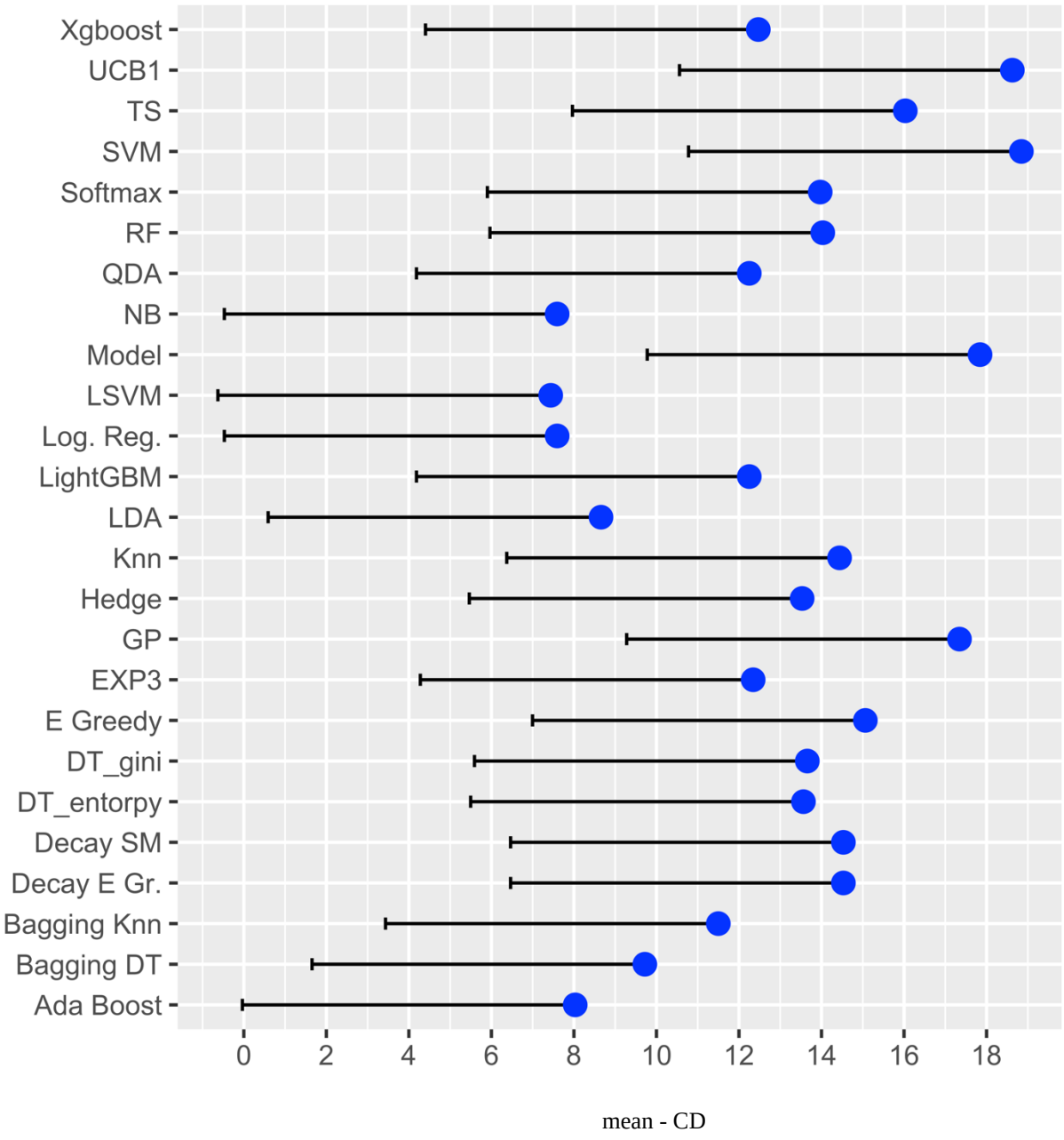


Figure 6.7: Comparison of the f1 score of all classifiers against each other with the Nemenyi test. Horizontal lines show the critical difference away from proposed pruning methods and any other methods. Groups of classifiers that are not significantly different (p = 0.0.5) are out of the lines from proposed methods

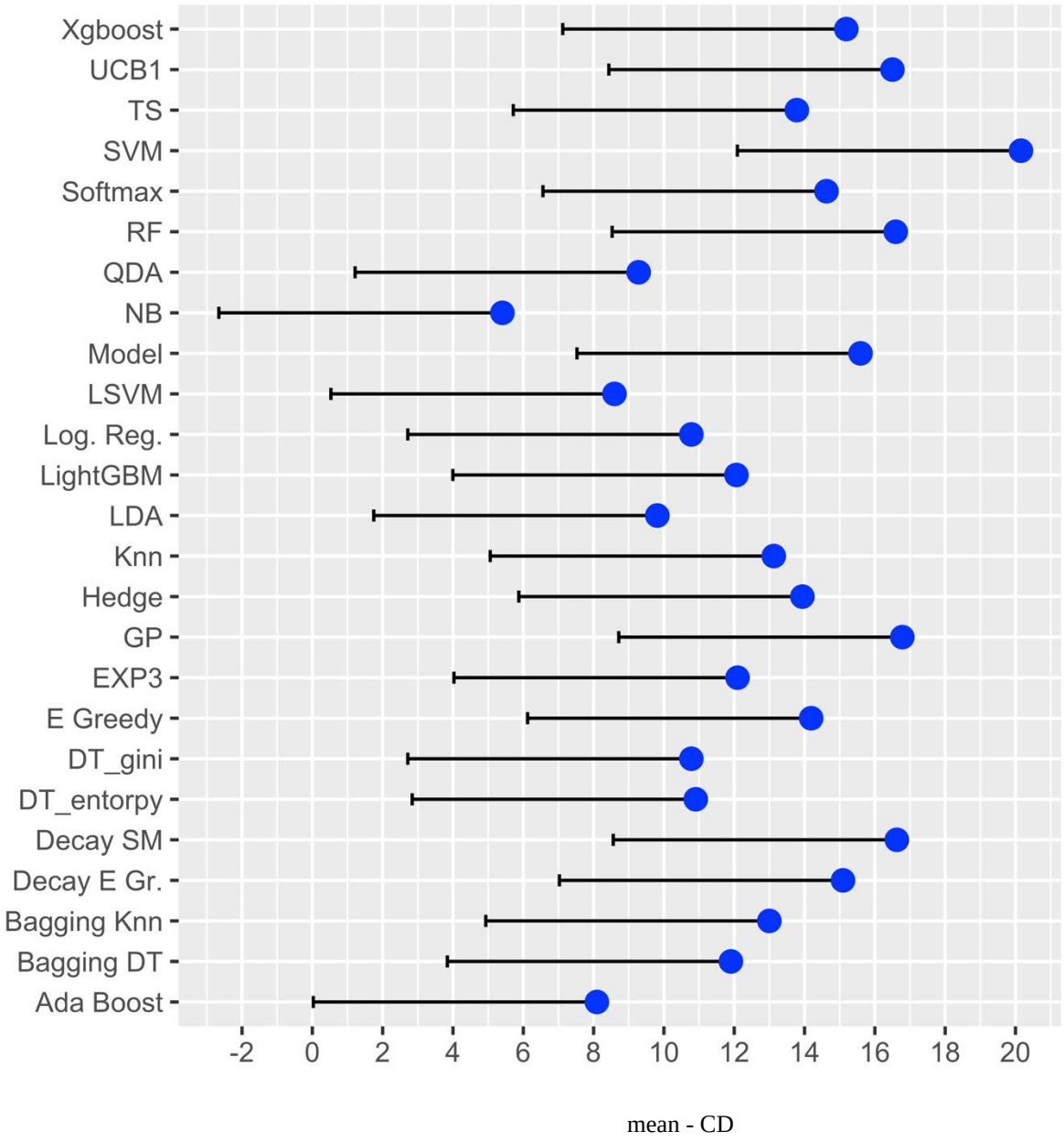


Figure 6.8: Comparison of the precision of all classifiers against each other with the Nemenyi test. Horizontal lines show the critical difference away from proposed pruning methods and any other methods. Groups of classifiers that are not significantly different (p = 0.0.5) are out of the lines from proposed methods.

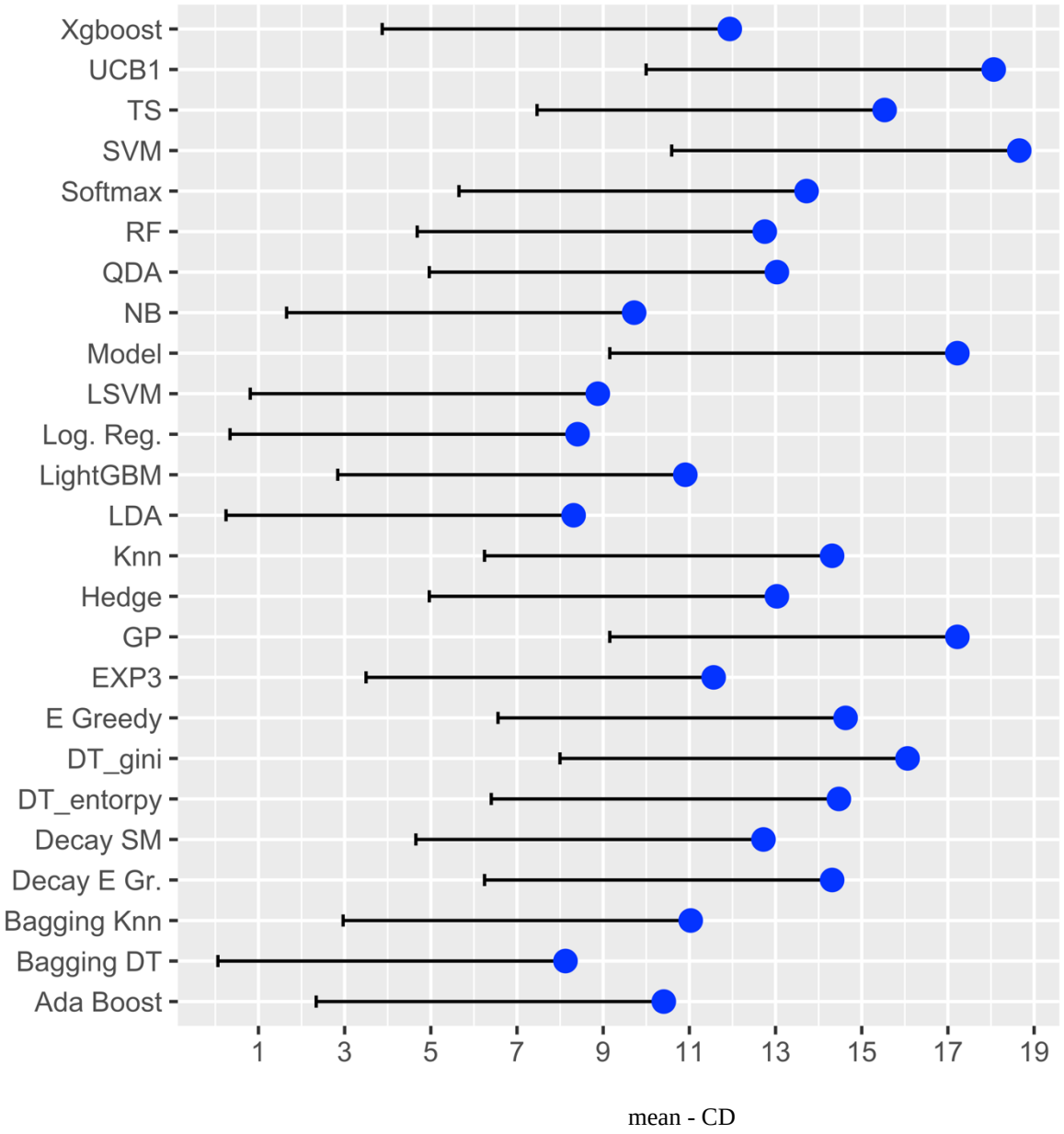


Figure 6.9: Comparison of the recall of all classifiers against each other with the Nemenyi test. Horizontal lines show the critical difference away from proposed pruning methods and any other methods. Groups of classifiers that are not significantly different (p = 0.0.5) are out of the lines from proposed methods.

6.2.2. Testing MAB Based Pruning on Deep Learning Networks

As described in Chapter 2, there are different architectures for deep learning which can be broadly categorised as feed forward networks, convolutional neural networks (ConvNets), recurrent neural networks (RNN). Several experiments were carried out within these categories with different data sets. The pruning algorithms described in Section 6.1 were used on the data sets as well as the following two methods which were adapted so they could be applied to removing neurons:

- A greedy algorithm, which is a generalised version of an algorithm presented in Polyak & Wolf [119], Hu et al. [120] and Luo & Wu [121] which we have adapted to prune neurons instead of feature maps. In this algorithm, compute the variance over the output of each neuron giving samples (four examples) of training data set then the neurons with the weak activation (activation has less variation) will be pruned.
- An algorithm that prunes based on magnitude (weights vector) [113] which was published recently and showed good results [134, 146]. The absolute magnitude of the weights vector for each neuron is computed. Then, neurons with the sum magnitude less than threshold is removed. The threshold value is specified by the user.

Table 6-10 summarises the results obtained, listing the algorithms applied, layer selected, number of neurons available for pruning, the methods used and the proportion pruned. The implementation code along with results are also available online[18].

.

[18] https://github.com/SalemAmeen/pruning_deep

Model/ Data set	Layer	No. of Neurons	Model	E Greedy	Decay E Greedy	Softmax	Decay Softmax	UCB1	TS	Hedge	EXP3	Magnitude	G. Prune	Aproxi. Neuron Pruned
MLP (Reuters)	2	512	0.786	0.79	0.79	0.79	0.79	0.79	0.79	0.79	0.79	0.75	0.74	50%
LeNet (MNIST)	3	128	0.98	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.97	0.95	62%
AlexNet (ImageNet)	6	4096	0.56	0.57	0.57	0.55	0.55	0.58	0.58	0.53	0.54	0.53	0.57	50%
AlexNet (ImageNet)	7	4096	0.56	0.58	0.58	0.56	0.56	0.57	0.57	0.56	0.56	0.55	0.54	50%
Conv (Cifar10)	5	512	0.81	0.82	0.82	0.82	0.82	0.82	0.82	0.82	0.82	0.81	0.8	21%
Conv (Cifar100)	5	512	0.43	0.45	0.45	0.44	0.45	0.45	0.45	0.44	0.44	0.43	0.41	20%
Conv (IMDB)	4	250	0.87	0.89	0.89	0.88	0.89	0.89	0.89	0.89	0.88	0.83	0.84	20%
Conv (SVHN)	5	512	0.96	0.97	0.97	0.97	0.97	0.97	0.97	0.97	0.97	0.9	0.92	39%
Siamese Graph (MNIST)	2	128	0.99	1	0.99	0.99	0.99	1	1	0.99	0.99	0.93	0.93	50%
LSTM (IMDB)	2	128	0.8	0.81	0.81	0.81	0.81	0.81	0.81	0.81	0.81	0.79	0.78	23%
Bi. LSTM_1 (IMDB)	2	64	0.82	0.84	0.83	0.83	0.83	0.84	0.84	0.83	0.8	0.8	0.81	78%
Bi. LSTM_2 (IMDB)	3	64	0.82	0.83	0.83	0.83	0.83	0.83	0.83	0.83	0.8	0.8	0.8	78%
EtoE Mem (bAbI)	4	32	0.86	0.87	0.88	0.87	0.87	0.88	0.87	0.87	0.88	0.81	0.83	40%
Hierar. RNN (MNIST)	2	128	0.96	0.97	0.95	0.98	0.97	0.98	0.97	0.98	0.98	0.9	0.92	15%

Table 6-10: The result based on the accuracy of pruning deep neural networks on different data sets using different architectures. The table shows the pruned layer, number of neurons in pruned layer and the percentages of removed neurons in the layer. The green cells indicate that the method has good accuracy in contrast of red cell. The arrows point up if the error high, down if it is low or in right direction if it is in between.

The remainder of this section elaborates upon the results presented in Table 6-10:

- Subsection 6.2.2.1 summarises how the results from applying the pruning methods MLP were obtained.
- Subsection 6.2.2.2 summarises how the results from applying the pruning methods on ConvNets were obtained.
- Subsection 6.2.2.3 summarises how the results from applying the pruning methods on RNN were obtained.
- Subsection 6.2.2.4 presents the results of applying the Friedman and Nemenyi test to compare the performance of the methods.

Each of the subsections is structured so that it presents: (i) the data set used, (ii) hyperparameters that used for the methods (iii) MAB algorithms applied (iv) comparison with other methods.

6.2.2.1. Pruning Multilayer Perceptron (MLP)

Data set. The data set used contains structured information about newswire articles that can be assigned to five classes and is known as the Reuters newswire topic classification task [200]. The main reason for using this collection is that it is one of the most classic collections for benchmarking text classification and allows comparison with previous studies. This data set includes 8,982 documents for training and 2,246 for testing.

Hyperparameters. A MLP (feed forward neural networks) with 512 neurons in one hidden layer is used. We found that the model was prone to overfitting, and we therefore experimented with introducing a dropout layer (with 0.5 probability of dropping neurons) after the hidden layer. The initial learning rate for Adam was 0.0001, and we found that the performance was much less sensitive to the learning rate with Adam than with other GD optimizations. Adam's parameters were set to 0.9 and to 0.99 for α and β respectively. The maximum number of words was set to 1,000 and the batch size[19] was set to 32. We trained the model for 50 epochs.

[19] In the neural network terminology: one epoch = one forward pass and one backward pass of all the training examples. batch size = the number of training examples in one forward/backward pass.

MAB algorithm. After the model was trained, we started applying the MAB pruning. We pruned the first fully connected layer that has 512 neurons. The proposed algorithms prune 150 neurons out of 512.

Comparison to other work. From Table 6-10, we draw two conclusions:

- All proposed algorithms can prune 50% of the model without any loss in performance. In contrast, greedy pruning and pruning based on magnitude show a decrease in performance when the same number of neurons have been pruned as the MAB based algorithms.
- Pruning based on magnitude has the least computation time but the computation time of the greedy prune algorithm is O(4*512F) while the computation time[20] of MAB is O(1200F) where F is the time for forward propagation. Forward propagation can vary between two algorithms and we ignored the other computation like assigning the rewards and updating other parameters, which can work in parallel with forward propagation. In addition, *F* can be represented by two forward propagations, one to compute the loss before pruning and the other after pruning.

6.2.2.2. Pruning ConvNets model

The ConvNets architecture has been widely used and this section presents, describes the experiments using many of the existing models starting from LeNet model trained on the MNIST data to AlexNet trained on ImageNet [21].

6.2.2.2.1. LeNet model

Data set. The MNIST [83] data set was used and is a collection of handwritten digits which contains 60,000 examples of training data and 10,000 examples of test data. The digits have been size-normalized and cantered in 28 by 28 pixel grey scale images.

Hyperparameters. LeNet (Lenet-5) [83] is a convolutional network with two convolutional layers and one dense layer. The network achieves 98% accuracy on MNIST [83]. We found that the model is prone to overfitting, and experimented with introducing a dropout layer

[20] Does not include playing each neuron once

(with 0.5 probability of dropping neurons) after each fully connected hidden layer. The learning rate for adadelta [201] was 0.01, ρ=0.95 and epsilon=0.00000001. The model trained with a batch size of 128 and with number of epochs set to 12.

MAB algorithm. After the model was trained, we started applying the MAB pruning algorithms to prune the first fully connected layer, as it has the most neurons (128 neurons). with play time set to 500.

Comparison to other work. From Table 6-10, all the proposed algorithm can prune 62% of the model without any loss in the performance. Our proposed algorithms outperform the original unpruned model and outperform the greedy and pruning based on magnitude algorithms.

6.2.2.2.2. AlexNet Model

Data set and Hyperparameters. The ImageNet ILSVRC-2012 data set [21, 22] was used, which has 1.2M training examples and 50k validation examples. We use a pre-trained AlexNet Caffe model [110], which has 61 million parameters across 5 convolutional layers and 3 fully connected layers. The AlexNet Caffe model achieved a top-1[21] accuracy of 56% and this model took 75 hours to train on an NVIDIA Titan X GPU [134].

MAB algorithm. We used pruning based MAB algorithms to prune 50% of both FC6 (layer 6) and FC7 (layer 7). These layers were chosen because they have the most neurons. The play time was set to 9,000.

Comparison to other work. Table 6-10 shows the results and we notice the following:

- **FC6**: MAB pruning based on UCB1 and Thompson Sampling improves the result by 0.03% and Epsilon-Greedy and decay Epsilon-Greedy improve it by 0.02. Pruning based Softmax, Hedge and EXP3 show some decline in the performance over the original model. Greedy pruning shows an improvement in this layer as much as Epsilon-Greedy does while pruning neurons based on magnitude has the worst performance.

[21] In ImageNet Challenge, the accuracy computed based on the top-1 prediction and top-5 where: Top-1 number is how many times the correct label has the highest probability predicted by the network. Top-5 number is how many times the correct label is within the top 5 classes predicted by the network.

- **FC7**: MAB pruning based on Epsilon-Greedy and decay Epsilon-Greedy show better results than the others with an improvement on the original model by 0.03%. UCB1 and Thompson Sampling show good results and improvement over the original unpruned model. In contrast to greedy pruning, pruning based on the magnitude does not perform well.

6.2.2.2.3. ConvNet on cifar-10 data set

Data set. The cifar-10 data set [202] is used and is composed of 10 classes of natural images with 50,000 training images, and 10,000 testing images. Each image is an RGB image of size 32x32. For this data set, we pre-process the data using global contrast normalization and ZCA whitening as was used by Goodfellow et al. [96]. We use the last 10,000 images of the training set as validation data.

Hyperparameters: This ConvNet has four convolutional layers and two fully connected layers trained on cifar-10 [202]. The network achieves 81% accuracy on cifar-10. We find that the model is prone to overfitting, and we experimented with introducing a dropout layer with 0.5 probability of dropping neurons after each fully connected layer and with 0.25 probability of dropping neurons after every convolution hidden layer. The learning rate of GD was 0.01, momentum was 0.9 and decay was 0.000001. In addition, the batch size was set to 32. We trained the model for 200 epochs.

MAB algorithm. After the model was trained, we started applying MAB pruning algorithms to prune the first fully connected layer, as it has the most neurons. The play time was set to 1200.

Comparison to other work. From Table 6-10, all proposed algorithm can prune 21% of the model without any loss in the performance. Our proposed algorithms outperform the original unpruned model and outperform the greedy and pruning based on magnitude algorithms.

6.2.2.2.4. ConvNets on cifar100 data set

This experiment used the cifar-100 [203] data set, which is the same size and format as the cifar-10 data set, but contains 100 classes, with only one tenth as many labelled examples per class. We use the same hyperparameters on cifar-100 that same we found to work well on cifar-10 in the previous subsection. We obtained a test set accuracy of 43%. The algorithm

could prune 103 neurons until the performance started to decrease. All the MAB algorithms show better results over the original unpruned networks and other pruning techniques.

6.2.2.2.5. ConvNets on SVHN data set

Data set. The SVHN data set [204] is a collection of 32x32 colour images. The data set is acquired from house numbers in Google Street View images. There are 73,257 digits in the training set and 26,032 digits in the test set. The task of this data set is to classify the digit located at the centre of each image. Pre-processing of the data set follows Goodfellow et al. [96], who used local contrast normalization.

Hyperparameters: We followed the same approach as on cifar-10 [202], where the model has four convolutional layers and two fully connected layers, to achieve 96% accuracy. The only difference is that we used a batch size of 128 and 20 epochs to speed up the training time.

MAB algorithm. After the model was trained, we applied the MAB pruning algorithms to prune the first fully connected layer, as it has the most neurons. With a play time of 1,200, the proposed algorithms can prune nearly 205 neurons as shown in Table 6-10.

Comparison to other work. From Table 6-10, all proposed algorithm can prune 39% of the model without any loss in the performance. Our proposed algorithms outperform the original unpruned model and outperform the greedy and pruning based on magnitude algorithms.

6.2.2.2.6. ConvNets on IMDB data set

Data set. This experiment used the IMDB (Internet Movie Database) data set [205], which is a movie reviews data set classifying positive or negative sentiments about reviews. It consists of 50,000 labelled movie reviews. The 100,000 movie reviews are divided into two data sets. 25,000 labelled training instances, and 25,000 labelled test instances. There are two types of labels: Positive and Negative. These labels are balanced in both the training and the test set. Several authors have used this data [78, 206, 207] to classify positive or negative sentiments about reviews.

Hyperparameters. We used ConvNet with the following architecture:

- Embedding layer is the first layer of ConvNets to map the vocabulary indices to vectors [208] (map the words to vectors). This is a technique where words are encoded

as real-valued vectors in a high dimensional space, where the similarity between words in terms of meaning translates to closeness in the vector space

- The first layer is followed by 1D convolution layer and fully connected layer with 250 neurons.

We found that the model is prone to overfitting, and therefore experimented with introducing a dropout layer with 0.2 probability of dropping neurons after the embedding layer and fully connected layer but not the convolution layer. The learning rate for Adam was 0.01 with $\alpha = 0.9$ and $\beta = 0.99$. The maximum number of features is 5,000, maximum lengths is 400, batch size is 32, embedding dimension is 50, number of filters is 250, filter length is 3 and hidden dimension is 250.

MAB algorithm. After the model was trained, the MAB pruning algorithms were applied with play time set to 800 and with the same hyperparameters given in subsection 6.2.1. The MAB pruning algorithms were used to prune the first fully connected layer, as it has the most neurons. The results improved when approximately 50 neurons were pruned Table 6-10. Using 1D convolution layer which has less weights than 2D make the neurons in the following layer more important.

Comparison to other work. From Table 6-10, we draw two conclusions:

- All proposed algorithm can prune 20% of the model without any loss in the performance. In contrast, the performance of the greedy pruning algorithm begins to decrease earlier than the MAB based algorithms.
- Pruning based on magnitude has improved the performance as well.

6.2.2.2.7. Siamese Graph

Data set. The MNIST [83] data set which is described above is also used for this experiment.
Hyperparameters. To train the Siamese graph [209], we followed Hadsell et al. [209] by computing the Euclidean distance on the output of the shared network and optimizing the loss function. We experimented with using dropout with a 0.1 probability of dropping neurons after each fully connected hidden layer. The learning rate for RMSProp was 0.01, ρ=0.95 and epsilon=0.00000001. The model was trained with a batch size of 128 and with 20 epochs.

MAB algorithm. After the model was trained, we started applying the MAB pruning algorithms to prune 128 neurons with 133,504 shared parameters between two branches, as it has the most neurons (128 neurons). The play time was set to 500 and the proposed algorithms can prune 50% neurons as shown in Table 6-10.

Comparison to other work. From Table 6-10, when half of the model is pruned, pruning based on MABs have not decrease the accuracy in contrast to the other pruning algorithms.

6.2.2.3. Pruning RNN

We also applied the MAB pruning algorithms to the RNN family of networks. Instead of pruning neurons in a fully connected layer as in the previous subsections, here we prune neurons in recurrent layers. The following subsections describe the experiments with four well-known architectures of RNN.

6.2.2.3.1. LSTM Model

Data set and Hyperparameters. First, we trained LSTM [100] on the IMDB data set [205] explained in subsection 6.2.2.2.6. The model contains an embedding layer as the first layer and then followed by an LSTM layer and then a fully connected layer. The experiments used 20000 features, the maximum length of texts is 80 and the batch size is 32. We also used dropout with a 0.2 probability of dropping neurons after first and second layer. The learning rate used for Adam was 0.001.

MAB algorithm and Comparison to other work. We applied MAB pruning algorithms with a play time of 400 to prune the LSTM layer. The performance of the pruned networks improved by around 2% when the MAB pruning algorithms pruned 30 neurons out of 128 as shown in Table 6-10.

6.2.2.3.2. Bidirectional LSTM

Data set and Hyperparameters. We trained a bidirectional LSTM [210] on IMDB data set. The embedding layer in the model has maximum features of 20000, maximum length is 100, batch size is 32 and number of epoch is 20. The model has an embedding layer, two LSTM layers and a fully connected layer. We used dropout with a 0.5 probability of dropping neurons. The learning rate used for Adam was 0.001.

MAB algorithm and Comparison to other work. The MAB pruning algorithms were applied with play time set to 180 to prune two LSTM layers. The performance in both layers increased until it pruned nearly 50 neurons out of 64, except when using EXP3 and two other pruning methods as shown in Table 6-10.

6.2.2.3.3. End-To-End Memory Networks

Data set. The Facebook bAbI data set [211] is a synthetic data set for testing a model's ability to retrieve facts and reason over them. In this data set, a given (question-answer) QA task consists of a set of statements, followed by a question whose answer is typically a single word (in a few tasks, answers are a set of words). The data set consists of 20 different tasks with various emphases on different forms of reasoning. For each question, only certain subsets of the statements contain the information needed for the answer, and the rest are essentially irrelevant distractors.

Hyperparameters. We trained End-To-End Memory Networks [101] on bAbI data set [211] with the specifications of 20000 maximum features, maximum length is 80 and the batch size is set to 32. The model has three embedding layers followed by an LSTM layer and a fully connected layer. We use dropout with 0.5 probability of dropping neurons after each layer except the output. The learning rate used for RMSProp was 0.0001.

MAB algorithm and Comparison to other work. We pruned the LSTM layer using different MAB pruning algorithms with a play time of 400. However, there are lots of extra neurons in this layer and only a few neurons can do the same work as the full neurons where 40% out of total 32 neurons. MAB pruning algorithms show good results compared to the original pruned model and the other two pruning algorithms as shown in Table 6-10.

6.2.2.3.4. Hierarchical RNN

Data set and Hyperparameters. We used the hierarchical RNN (HRNN) [212, 213] to train a model for the MNIST data set. The model has a time distributed layer, two LSTM layers, and a fully connected layer. At the first LSTM layer, every 28 by 1 column is encoded to 128 vectors, while at the second LSTM layer, all the 28 columns are encoded in 28 x 128 to represent the whole image. Then, the fully connected layer is used to make a prediction. The RMSProp optimizer was used to train the model with a batch size of 32, 10 classes and 10 epochs.

MAB algorithm and Comparison to other work There was some improvement in the model after pruning except for annealing Softmax where the performance declined compared to the original model when 15% of the model is pruned as shown in Table 6-10.

6.2.2.4. Comparison using the Friedman and Nemenyi Tests

To compare the results from the different algorithms, the Friedman test has been applied between the eight different proposed algorithms, the original unpruned model and the other two pruning algorithms. The p value is 5.597×10^{-17}, which is less than 0.05 and indicates there is a significant difference between the different algorithms. Table 6-11 shows the average difference between the algorithms where the higher numbers are better because the results are based on the accuracy. Table 6-11 shows that all the proposed MAB pruning algorithms are better than the original model. Pruning based on UCB1 and Thompson Sampling are the best. To assess whether there is significant difference between all pruning algorithms and the original unpruned model, the Nemenyi test is used and the results from the test are presented in the Figure 6.10. Pruning based on UCB1, Thompson Sampling, Epsilon-Greedy and decay Epsilon-Greedy statistically improve the model and these improvements are not based on chance, as shown in the Figure 6.10. In addition, pruning based on decay Softmax, Hedge and Softmax statistically improved over pruning based on greedy pruning and the pruning based on magnitude. Full details of all tests are available online[22].

Name of Method	Mean Rank
UCB1	8.679
Thompson Sampling	8.142
Epsilon-Greedy	8.107
Decay Epsilon-Greedy	7.607
Decay SM	6.964
Hedge	6.714
Softmax	6.643
EXP3	6.036
Model	3.321
Greedy Prune	2.034
Less Magnitude	1.750

Table 6-11: Results of ranked accuracy results based on Nemenyi test, which is used to compare the different models on six different data sets.

[22] https://github.com/SalemAmeen/testing_python_friedman/blob/master/prune_neurons/DL/friedman.ipynb

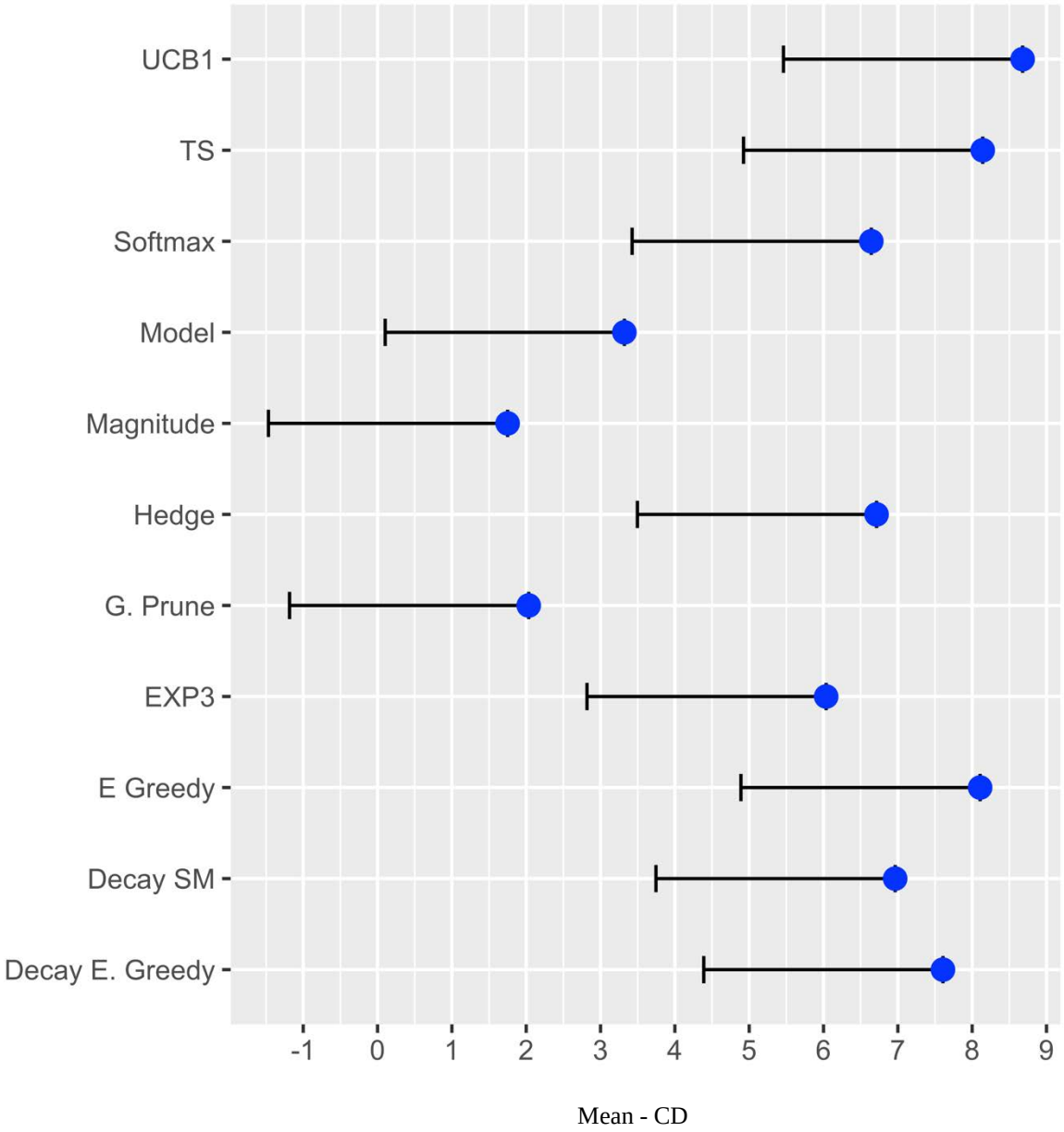


Figure 6.10: Comparison of all pruning algorithms against each other with the Nemenyi test. Horizontal lines show the critical difference away from proposed pruning algorithms, the orginal unpruned model and two other algorithms that are not significantly different (p = 0.0.5) are out of the lines from proposed algorithms. CD=3.218.

6.3. Discussion

The evaluation results presented in the previous sections indicate that the proposed pruning algorithms performed well on several data sets. Different sets of experiments were conducted

from which the main findings can be summarised as follows:

- In general, UCB1 and Thompson Sampling pruning algorithms show the best results among the proposed algorithms, especially when the play time is long enough. In general, UCB1, which is based on upper bound rewards, converge faster than the algorithm that samples the rewards from a Bernoulli distribution (Thompson Sampling). The bounded rewards reflect the unimportance of the model faster than the binary rewards.
- The adversarial bandit algorithms did not perform as well as the other MAB algorithms. One potential explanation for this is that adversarial bandits are developed primarily for situations where the reward is dynamic and changes with time while the other bandits assume stationary rewards.

6.4. Summary

This chapter has described applying MAB algorithms to prune neurons. The main algorithm was presented and three new bandit functions softmax, Hedge and EXP3 were used with some others explained in the previous chapter. The operation of these algorithms was illustrated using an example. The chapter presented an empirical evaluation in comparison to classification and regression methods using 16 data sets from UCI and Kaggle, showing that the MAB based methods were able to reduce the size of neural networks without reducing accuracy. An empirical evaluation on deep learning models trained on eight well known data sets also showed good results.

7. Multi-Armed Bandits for Pruning Feature Maps

This chapter, introduces a way to remove the least important feature maps from a convolutional layer of ConvNets to speed up inference time.

In ConvNets, while the fully connected layer has the most parameters, the convolutional layer is responsible for the most floating-point operations (*FLOPS*[23]) [115, 214]. To reduce the size of a ConvNets, the ideal way is to remove many neurons, as shown in the previous chapter, while removing feature maps will lead to speeding up the inference time of ConvNets.

Figure 7.1 shows a ConvNet before and after removing a filter $F_{i,j}$ and its corresponding feature map $X_{i+1,j}$. Molchanov et al. [115] have shown that the number of *FLOPS* for a ConvNet can be computed by:

$$FLOPS = 2{\times}H_i{\times}W_i{\times}(C_{in}{\times}K^2 + 1){\times}C_{out}$$

Where $C_{in}$and C_{out} are the number of inputs and outputs of feature maps (channels), H_i and W_i are the height and the width of the input feature map and *K* is the width or the length (assumed to be symmetric) of the kernel.

For example, in third layer in Table 2-1, the kernel is 3×3 (*K=3*), the number of input feature maps are 32 ($C_{in} = 32$), the number of the output is 64 ($C_{out} = 64$), and the height and width are both 15. The number of *FLOPS* is:

$$FLOPS = 2{\times}15{\times}15{\times}(32{\times}(3)^2 + 1){\times}64 = 8{,}323{,}200$$

[23] *FLOPS* is the commonly used measure to compare computation complexities of ConvNets

Removing one feature map in this layer leads to reducing C_{out} by 1 which leads to

$$FLOPs = 2\times15\times15\times(32\times(3)^2 + 1)\times63 = 8{,}193{,}150$$

In addition, C_{out} will be C_{in} for the next layer which leads to improving the $FLOPs$ in the following layer. The *FLOPS* of all convolutional layers in this model and some other models on different data sets are shown in Table 7-1. As this illustrates, the ability to prune feature maps, can also help improve the speed of inference of ConvNets without reduction in accuracy.

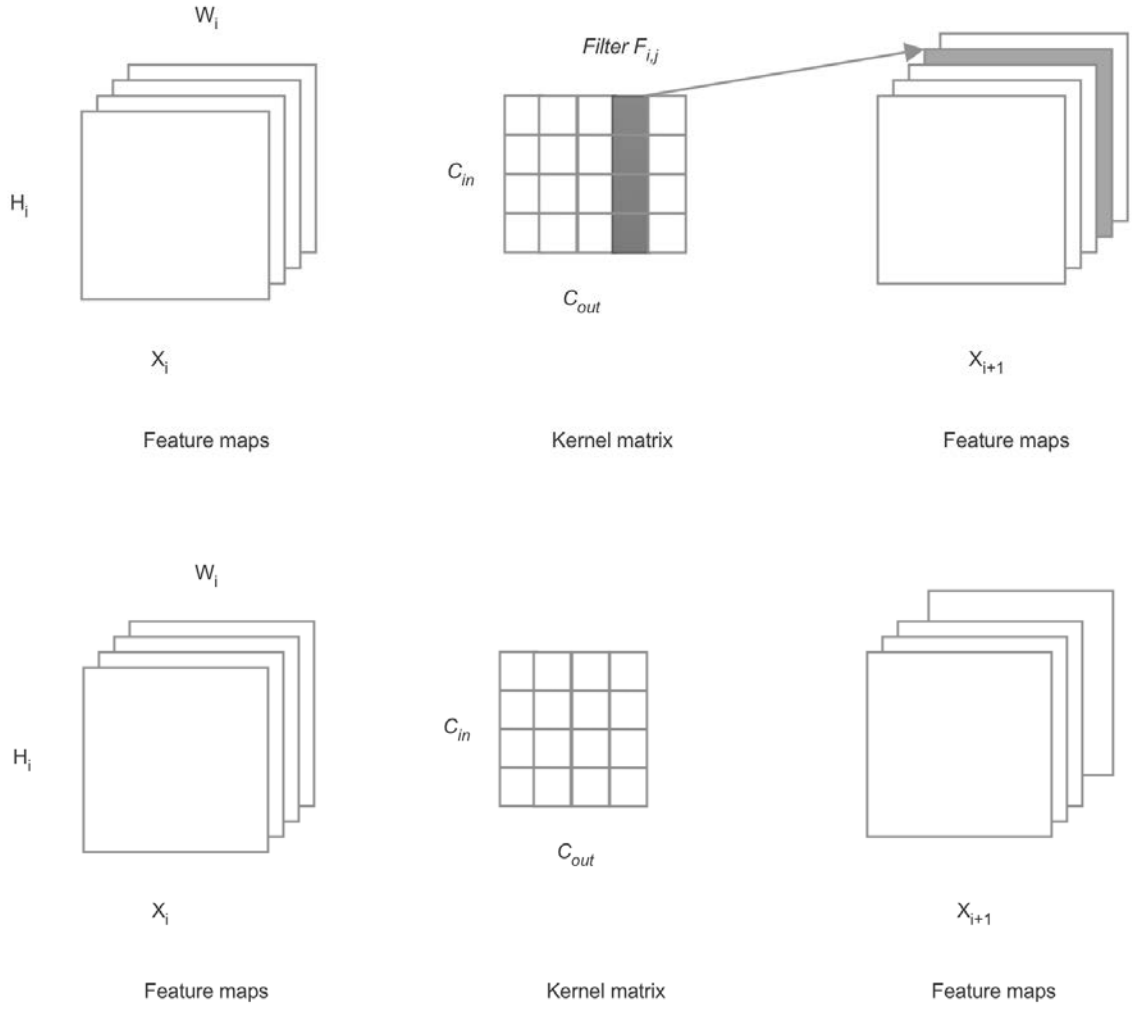


Figure 7.1 Removing the filter $\mathcal{F}_{i,j}$ and corresponding feature map in X_{i+1} in ConvNets. The top diagram shows the two layers before pruning while the bottom diagram shows the two layers after pruning the filter and feature map.

As described in Chapter 4, there are many MAB algorithms and a number of these were tried for removing weights and neurons and results presented in Chapters 5 and 6 respectively. The lessons learned from the evaluations presented in Chapters 5 and 6 were used to select the MAB algorithms to evaluate for removing feature maps, and UCB1 and Thompson Sampling were chosen, given that both showed good performance and good running time.

Model Name	**Layer**	**H_i**	**W_i**	**K**	**C_{in}**	**C_{out}**	***FLOPS***
LeNet (MNIST)	1	26	26	3	3	32	1211392
	2	24	24	3	32	32	10653696
Conv (Cifar10)	1	32	32	3	3	32	1835008
	2	30	30	3	32	32	16646400
	3	15	15	3	32	64	8323200
	4	13	13	3	64	64	12481664
Conv (Cifar100)	1	32	32	3	3	32	1835008
	2	30	30	3	32	32	16646400
	3	15	15	3	32	64	8323200
	4	13	13	3	64	64	12481664
Conv (SVHN)	1	32	32	3	3	32	1835008
	2	30	30	3	32	32	16646400
	3	15	15	3	32	64	8323200
	4	13	13	3	64	64	12481664

Table 7-1: Examples of computing *FLOPS*.

The rest of this chapter is organized as follows. The direct method is presented in Section 7.1. Experimental evaluations and analysis are presented in Section 7.2. The results are discussed in Section 7.3 and the chapter concludes with a brief summary in Section 7.4.

7.1. Direct Method

Molchanov et al. [115] define the direct method (oracle prune) as the optimal criterion to prune feature maps and its corresponding filters. It would be an exact empirical evaluation of each feature map, accomplished by removing each filter in turn and recording the difference of loss function. While the oracle is optimal for this greedy procedure [115], as

described in Chapter 3, it is prohibitively costly to compute [115]. Since the estimate of parameter importance is key to both the accuracy and the efficiency of this pruning approach, we evaluate two proposed pruning methods based on UCB1 and Thompson Sampling.

7.2. Evaluation

This section reports on the evaluation of two proposed MAB pruning algorithms, namely UCB1 and Thompson Sampling, when they are used to prune feature maps.

The objectives of the evaluation are the same as in Chapter 6 except that there is a comparison in relation to the direct method which is possible here because there are only a few feature maps compared to the number of weights and neurons.

This section is organised in two parts. First, Subsection 7.2.2 presents an initial experiment that aims to explore the extent to which the MAB methods produce results that are consistent with the direct method which is the oracle. This is done on just the MNIST data and LeNet. Subsection 7.2.3 presents the results obtained using MAB pruning algorithms with respect to the two pruning algorithms (greedy pruning and pruning neurons based on the magnitude).

Both the proposed MAB algorithms were implemented using the Python programming language. All the reported experiments in this chapter were conducted using NVIDIA TITAN X.

7.2.1. Data sets

The algorithms were evaluate using six data sets, which include four data sets described in Chapter 6, namely MNIST, Cifar10, Cifar100, SVHN and the following two additional data sets:

- Caltech-UCSD Birds 200-2011 data set: This data set [215] consists of nearly 6,000 training images and 5,700 test images, covering 200 species. Branson et al. [216] show that training ConvNets from scratch on the Birds-200 data set achieves test accuracy of only 10.9% and it gets improved by using transfer learning. The following steps are used to train AlexNet to get 71% accuracy on this data set:

1. First, AlexNet (that was described in chapter 6) is pre-trained on the 1.2 million images from the ImageNet data set and used as a feature extractor.
2. Then, the final 1,000-class AlexNet output layer is chopped off and replaced by a 200-class CUB-200-2011 output layer. The weights of the new layer are initialized randomly, and stochastic gradient descent (SGD) with 0.9 momentum, batch size 64, and back propagation are used to learn the weights of the new 200-class output layer while the weights of the old AlexNet are fixed. We use the learning rate of 0.001, weight decay 0.0001 and trained until 90 epochs.
3. Finally, the model is fine-tuned by training the entire network jointly with a small learning rate 0.0001 for 10 epochs.

- Oxford Flowers 102 data set: This data set [217] consists of nearly 2,040 training images and 6,129 test images from 102 species of flowers. The training procedure is the same as for Birds-200, except that AlexNet was fine tuned using 40 epochs with a learning rate 0.00001 to achieve a test accuracy of 80%.

7.2.2. Intial Comparison with the Direct Method

Figure 7.2 shows the results when the direct, UCB1, and Thompson Sampling methods are applied to prune feature maps from the LeNet model trained on the MNIST data. In addition, the cumulative rewards are bounded between [-1,1] to show the relation between learnt rewards and the real change of the loss computed by direct method.

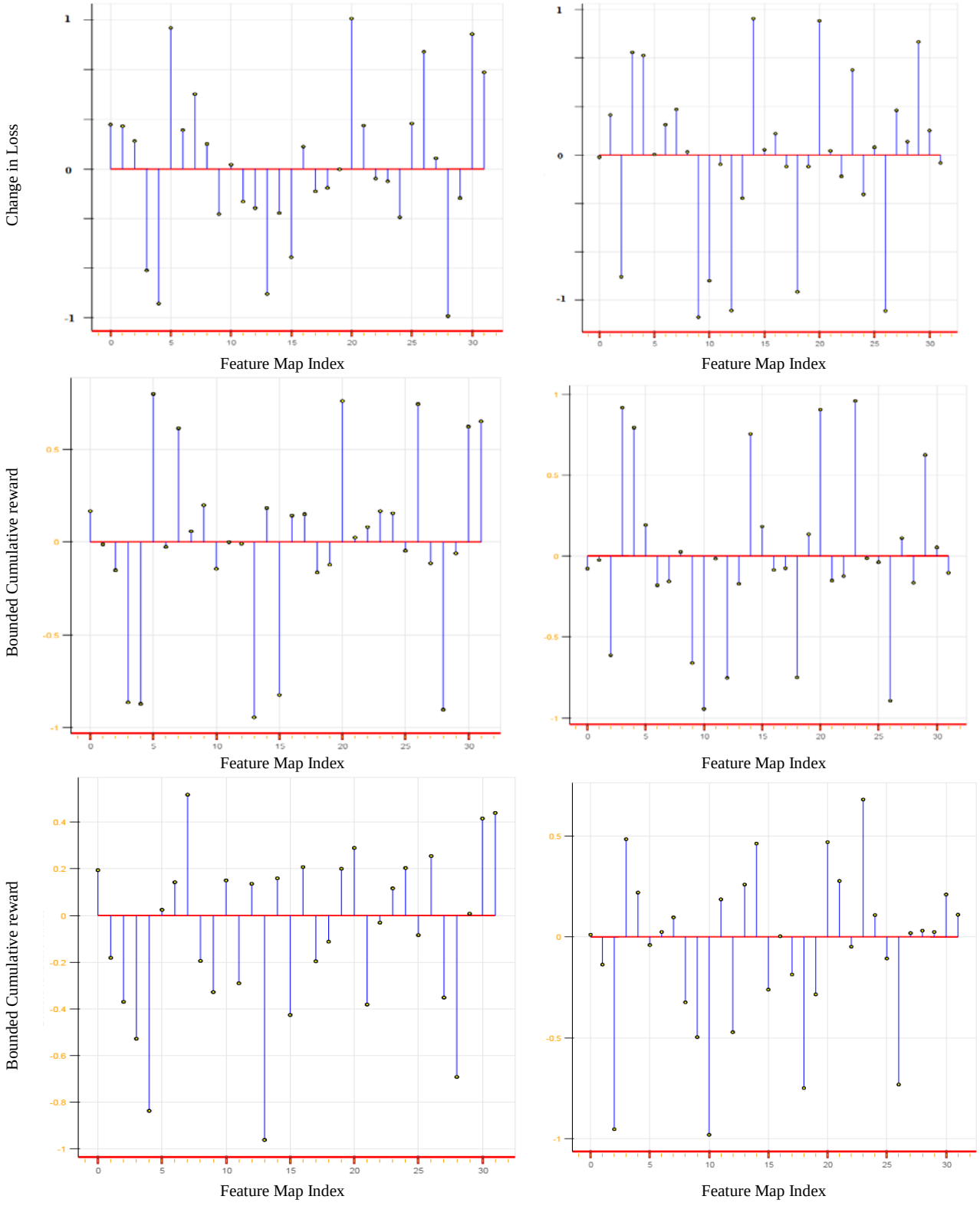


Figure 7.2: Change in training loss as a function of the removal of a single feature map from the LeNet model. The first convolutional layer is on the left and the second convolutional layer is on the right. The top row shows the results for brute force pruning, the middle is for UCB1 pruning and the bottom is for Thompson Sampling.

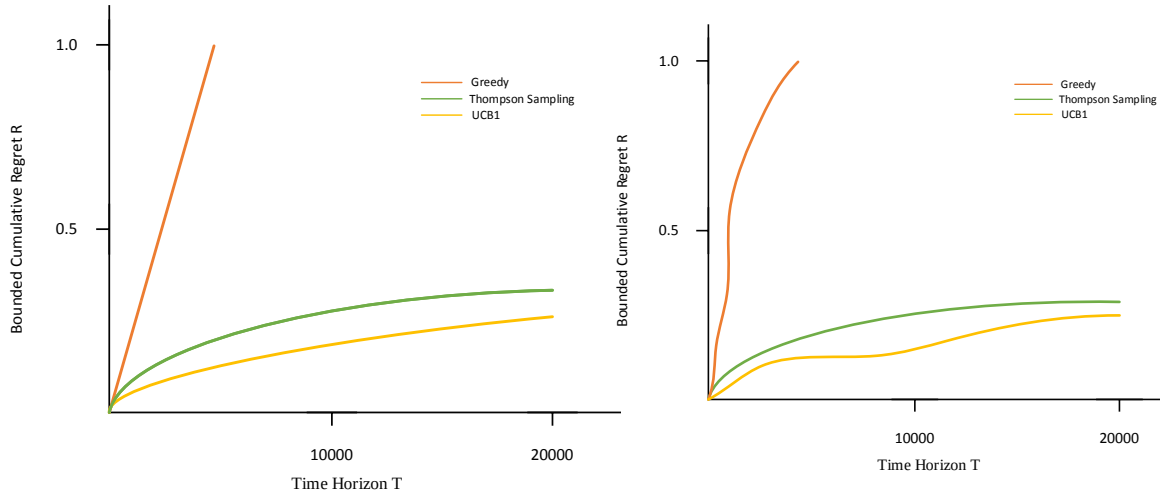
Figure 7.2 shows that both proposed algorithms result in high rewards for the feature maps that need to be pruned and low rewards for the ones that do not need to be pruned. There are some feature maps where the rewards are close to zero and where there is a difference from the direct method. For example, the direct method has a positive reward for the third feature map while UCB1 results in a negative reward. However, in practice, this is not significant given that the other feature maps have larger rewards.

The Pearson correlation coefficients between the direct method and the two MAB methods are:

- Correlation between direct method and UCB1 = 0.83
- Correlation between direct and Thompson Sampling = 0.80

As a further initial comparison, the regret of the algorithms was compared to a greedy algorithm. The greedy algorithm involved removing each feature map and evaluating the reward after presenting each example and comparing the selected feature map with the one selected by the direct method. This was repeated for 4,800 times and the cumulative regret computed, where for each play, a regret of one is allocated if the wrong (i.e., different from the direct method) feature map is selected and otherwise a zero is allocated if the correct feature map is selected. In the direct method, the horizon time was set to 60,000 examples for every feature map, which several authors have regarded as adequate to ensure optimality [111, 114, 115]. For the MAB algorithms, 20,000 random examples were used and unlike the other algorithms, not all the features are evaluated for each example given the nature of MAB algorithms. Figure 7.3 presents the results when the algorithms are applied on the two convolutional layers of the LeNet model.

These initial experiments on just the LeNet model trained on the MNIST data show promising results.



(a) The first convolution layer (b) The second convolution layer

Figure 7.3: Cumulative regret incurred on LeNet model trained on MNIST data set compared to direct method.

7.2.3. Pruning Feature Maps using UCB1 and Thompson Sampling

The previous subsection explored the extent to which MAB algorithms are able to identify the feature maps that should be deleted, where the direct method is used as the oracle. This was, however, limited to pruning the feature maps of the LeNet model after it had been trained on the MNIST data. This subsection presents the results of applying the algorithms on additional data sets.

Table 7-2 shows the results from applying the two MAB algorithms, a greedy pruning algorithm [119] and an algorithm that removes the filter that has the smallest absolute sum magnitude among the filters [146]. The first column in the table shows the model and the data set used to build the model. The second column shows which layer(s) have been pruned. For example, if the number is 2 then we prune the second layer. The term “all” in the second column means that all convolutional layers are considered. In other words, the number of arms is equal to the total number of feature maps in the ConvNets model. The remaining columns show the results before after pruning, including the approximate number of *FLOPS* before pruning (which is based on [108]) and the percentage reduction in the number of feature maps after pruning.

In all the experiments, the playing time is set to five times the number of feature maps (arms). For example, for pruning the convolutional layer in AlexNet trained on Bird-200, the total playing time is $5 \times 256 = 1{,}280$.

The results in Table 7-2, show that, in general, the UCB1 and Thompson Sampling pruning algorithms maintain the accuracy of the original unpruned models.

In addition, as the results in Table 7-2 show, the MAB methods outperform the other pruning techniques. The experiments were carried out layer by layer, starting with the first convolutional layer followed by the next layer and so on. We make two observations based on these results:

- First, that we expected a pattern where the later convolutional layers were more likely to be pruned than the earlier convolutional layers. We believed this would happen for two reasons for this. First, to recognize objects in images, the first layer aims to learn to recognize edges, the second layer combines edges to form motifs, the third learns to combine motifs into parts, and the next layer learns to recognize objects from the parts identified in the previous layer and so on [20]. From this sequence, the first layers will be for general feature detection while the later layers will aim to detect specific objects. Thus, it was expected that it would be easier for a pruning algorithm to determine which feature map does not belong to any classes in the later layers while that is difficult to remove the earlier feature maps given these layers extract the edges which relate to all classes. The second reason, is that in ConvNets, the later layers have a larger number of feature maps than the previous layers which makes it more likely to have feature map that are not important. However, the experiments show that this was true for only three of the four experiments with convolutional networks but not for AlexNet. Although further experimentation is needed, one reason for this might be that AlexNet was pre-trained on the ImageNet data and hence one would expect the need for pruning the earlier layers which may be too generic.
- Pruning all the convolutional layers together is better than pruning each layer separately. For example, when pruning the LeNet model based on all layers together the algorithm prunes nearly 22% from the total number of feature maps. We think the reason is that pruning each layer separately, we enforce each layer to prune some of the feature map while pruning feature maps in all layer, the pruning algorithm will determine the unimportant feature maps across the layers which is expected will be in later layers.

Model/ Data set	Layer	Before Pruning			After Pruning					
		No. Channels	Approx. FLOPS	Model	Greedy Prune	Based Magn	UCB1	Thompson Sampling	No. Channels	% Removed
LeNet (MNIST)	1	32	1M	↑ 0.98	↑ 0.98	↓ 0.91	↑ 0.99	↑ 0.99	28	12.5
	2	32	11M	↑ 0.98	↑ 0.97	↓ 0.9	↑ 0.98	↑ 0.98	26	18.75
	all	64	12M	↑ 0.98	↑ 0.98	↓ 0.91	↑ 0.99	↑ 0.99	50	21.88
Conv (Cifar10)	1	32	2M	↑ 0.81	↑ 0.8	↓ 0.73	↑ 0.81	↑ 0.81	28	12.5
	2	32	17M	↑ 0.81	↑ 0.8	↓ 0.74	↑ 0.81	↑ 0.81	24	25
	3	64	8M	↑ 0.81	↑ 0.8	↓ 0.73	↑ 0.81	↑ 0.81	52	18.75
	4	64	12M	↑ 0.81	↑ 0.8	↓ 0.73	↑ 0.81	↑ 0.81	50	21.88
	all	192	39M	↑ 0.81	↗ 0.8	↓ 0.74	↑ 0.82	↑ 0.82	140	27.08
Conv (Cifar100)	1	32	2M	↑ 0.43	↗ 0.4	↓ 0.31	↑ 0.43	↑ 0.43	30	6.25
	2	32	17M	↑ 0.43	↑ 0.41	↓ 0.3	↑ 0.43	↑ 0.43	28	12.5
	3	64	8M	↑ 0.43	↑ 0.42	↓ 0.35	↑ 0.43	↑ 0.43	56	12.5
	4	64	12M	↑ 0.43	↗ 0.41	↓ 0.35	↑ 0.43	↑ 0.43	54	15.63
	all	192	39M	↑ 0.43	↗ 0.42	↓ 0.35	↑ 0.44	↑ 0.44	148	22.92
Conv (SVHN)	1	32	2M	↑ 0.96	↗ 0.94	↓ 0.89	↑ 0.96	↑ 0.96	28	12.5
	2	32	17M	↑ 0.96	↗ 0.94	↓ 0.88	↑ 0.96	↑ 0.96	28	12.5
	3	64	8M	↑ 0.96	↑ 0.95	↓ 0.9	↑ 0.96	↑ 0.96	54	15.63
	4	64	12M	↑ 0.96	↑ 0.95	↓ 0.9	↑ 0.96	↑ 0.96	50	21.88
	all	192	39M	↑ 0.96	↗ 0.95	↓ 0.91	↑ 0.97	↑ 0.96	140	27.08
AlexNet (Flower-102)	1	96	105M	↑ 0.8	↘ 0.76	↓ 0.74	↑ 0.8	↑ 0.8	90	6.25
	2	256	223M	↑ 0.8	↘ 0.77	↓ 0.76	↑ 0.8	↑ 0.8	220	14.06
	3	384	149M	↗ 0.8	↘ 0.78	↓ 0.76	↑ 0.81	↑ 0.82	340	11.46
	4	384	112M	↗ 0.8	↘ 0.78	↓ 0.76	↑ 0.82	↑ 0.82	338	11.98
	5	256	74M	→ 0.8	↓ 0.78	↓ 0.77	↑ 0.83	↑ 0.82	228	10.94
	all	1376	663M	→ 0.8	↓ 0.78	↓ 0.77	↑ 0.83	↑ 0.82	980	28.78
AlexNet (Birds-200)	1	96	105M	↑ 0.71	→ 0.68	↓ 0.64	↑ 0.71	↑ 0.71	90	6.25
	2	256	223M	↑ 0.71	→ 0.68	↓ 0.65	↑ 0.71	↑ 0.71	222	13.28
	3	384	149M	↑ 0.71	↘ 0.69	↓ 0.67	↑ 0.72	↑ 0.71	344	10.42
	4	384	112M	↗ 0.71	↘ 0.7	↓ 0.69	↑ 0.72	↗ 0.71	334	13.02
	5	256	74M	↗ 0.71	↘ 0.7	↓ 0.69	↑ 0.72	↑ 0.72	226	11.72
	all	1376	663M	→ 0.71	↓ 0.7	↓ 0.7	↑ 0.72	↑ 0.72	978	28.92

Table 7-2: Result of pruning convolutional layers. The green cells indicate that the method has good accuracy in contrast of red cell. The arrows point up if the error high, down if it is low or in right direction if it is in between.

To assess if the differences between the algorithms are significant, the Friedman test was applied. The results indicated that there was a significant difference between the algorithms as the p values of Friedman test is 4.25×10^{-23}. Table 7-3 presents the average rank of the algorithms for pruning feature maps.

Name of Method	Mean Rank
UCB1	4.28
Thompson Sampling	4.10
Model before pruning	3.55
Greedy Pruning	2.03
Based on the Magnitude	1.03

Table 7-3: Average rank of the algorithms for pruning feature maps based on accuracy, where a higher rank is better.

Table 7-3 indicates that pruning feature maps with UCB1 has the best mean rank over the other algorithms followed by Thompson Sampling, while pruning filters based on the magnitude has the worst rank followed by greedy pruning of the weights.

Since the Friedman test shows a significant difference between the different methods, a post hoc test was used to find which algorithm(s) performed significantly better than the others. For this, we used the Nemenyi post hoc test and the result is shown in Figure 7.4 where the lines from the proposed algorithms with length of critical difference CD is plotted to show the significant difference to the proposed algorithms. The x-axis in the diagram is the axis on which we plot the average ranks of algorithms, where the rank increases from left to right. In Figure 7.4, the results indicate that the proposed algorithms performed statistically better than the greedy pruning and pruning filters based on the magnitude algorithms, as the difference between them is greater than the CD=1.133. However, these results do not allow us to reject the null hypothesis between the proposed pruning algorithms and the original unpruned model. The results and implementation of these tests are available online[24].

[24] https://github.com/SalemAmeen/testing_python_friedman/tree/master/pruning_fearturemap/Inbound

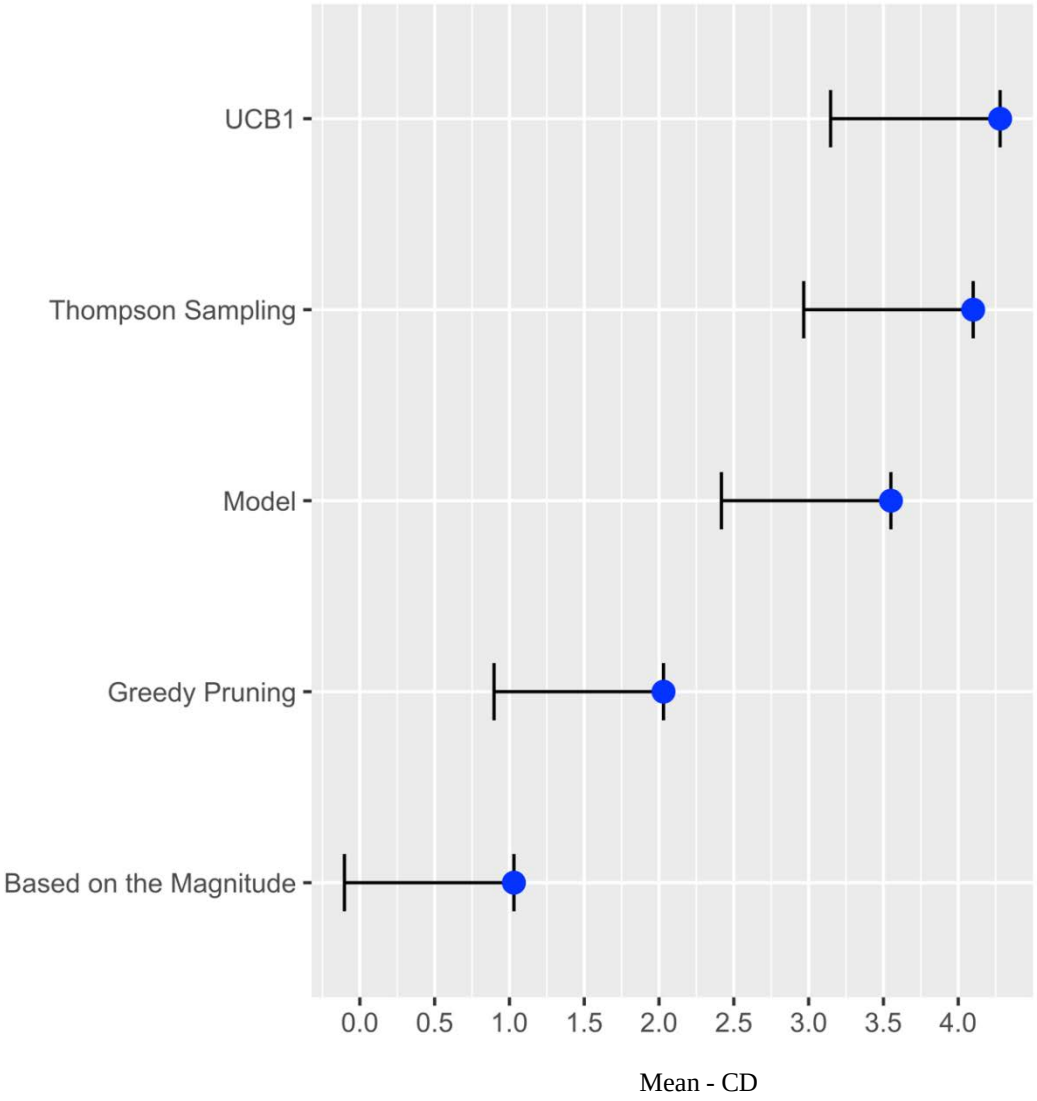


Figure 7.4: Comparison of all classifiers against each other with the Nemenyi test. Horizontal lines show the critical difference away from proposed pruning algorithms and any algorithms. CD=1.133.

7.3. Discussion

The evaluation results presented in the previous sections indicate that the two proposed pruning algorithms performed well on several data sets. Different sets of experiments were conducted from which conclusions can be drawn, as follows:

- MAB algorithms offer a useful way of pruning feature maps and corresponding filters

in ConvNets.

- Pruning based on UCB1 and Thompson Sampling mimic the direct method of pruning feature maps on the LeNet model and therefore have potential for being as effective as the direct method but without the computational overheads of the direct method.
- Pruning all convolutional layers together is better than pruning every layer separately. The pruning algorithm can determine which layer has the most unimportant feature maps.

7.4. Summary

We have developed two pruning algorithms that can prune convolutional layers in trained ConvNets model. The proposed pruning algorithms are based on UCB1 and Thompson Sampling. Our evaluation shows strong performance compared to the baseline approaches of greedy pruning or pruning based on the magnitude. One of the limitations of the proposed algorithms is pruning and testing one feature map at each play time, which makes the proposed algorithms slower than some baseline algorithm. In Chapter 8, we describe another pruning algorithms that include pruning multiple feature maps at the same time, reducing the total play time.

8. Pruning Multiple Neurons and Feature Maps using MABs

When a neural network is built with several neurons in its hidden layers, ideally each neuron should operate as an independent feature detector [103]. If two or more neurons begin to detect the same feature repeatedly (known as co-adaptation), the network is not utilising its full capacity efficiently given that it computes the activations for redundant neurons. Hinton et al. [103] provide one approach to counter the negative effects of such neurons by using a technique known as dropout [135][25], which essentially omits a random bunch of neurons during training to prevent co-adaptation [103]. Reed [111] and Wolfe et al. [114] show that this problem of neurons that interact or cancel each other can happen even after training the neural networks with dropout [114].

In this chapter, we introduce two pruning algorithms based on MABs to prune multiple neurons or feature maps at the same time. Each algorithm computes the cumulative average reward based on the change in the loss function after pruning multiple neurons or feature maps. The cumulative reward is then used to prune the neurons or feature maps after the playing time is finished.

The rest of this chapter is organized as follows. The expected advantage of pruning multiple neurons over one neuron at a time is present in Section 8.1. The proposed algorithms are presented in Section 8.2. Experimental evaluations and analysis are presented in Section 8.3 and Section 8.4 summarises the chapter.

[25] In addition, it is a way to prevent neural networks from overfitting.

8.1. The Advantage of Pruning Multiple Neurons over Pruning One

In term of co-adaptation, the main drawback of pruning one neuron, either based on a specific layer or throughout the layers, is that the pruning algorithm will prune the neuron that affects the change of loss. In other words, this algorithm assumes that any neuron effects the output without any interconnections with other neurons, which is the optimal case where the co-adaption is removed during training. However, the optimal case is hard to achieve and there is no guarantee that a model will be optimal Hinton et al. [103]. That is, algorithms that focus on removing single neurons, do not address the problem of detecting neurons that cancel each other out or interact in a way that has a negative impact on performance [111, 114]. However, the literature review in this study identified only the one method, namely OBS, that addressed this problem once a model has been trained. However, as described in Section 3.2.1.2, OBS is computationally demanding, as it needs to invert a Hessian matrix. Hence, the next section presents algorithms that aim to prune multiple neurons which may be in the same layer or across different layers.

8.2. MAB Algorithms for Pruning Multiple Neurons and Featue Maps

The basic idea for pruning multiple neurons and feature maps is similar to that described in Chapters 6 and 7 except that there are two main differences. First, a multi play MAB algorithm is used to select and remove multiple neurons or feature maps instead of one neuron or feature map. Secondly, the reward from removing a set of neurons (or feature maps) needs attributing to individual neurons (feature maps). This is done as follows.
Assume that x_i is the output of neuron i before it is pruned and δL_g is the change of the output after pruning multiple neurons (including neuron i) then, the reward for the neuron i will be given by,

$$r_i = x_i * X^g, \qquad (8.1)$$

Where X^g is the reward as a result of removing the neurons. Here, we also assume the neural networks use the ReLU function, which has been widely used in deep learning [218], where its output is positive.

Equation 8.1 has several properties that make it suitable a measure of reward:

- When there is no improvement from removing neurons or feature maps then the reward is zero.
- If the new model results in an improvement, then an overall reward X^g for the group of chosen neurons or feature maps will be generated. The extent to which a neuron contributes to the output (and hence the reward X^g) is dependent on the output of the neuron [120]. That is, the larger the output from a neuron, the more it will contribute to the output of the network. Hence the above measure allocates the reward in proportion to the output of a neuron.

Figure 8.1 presents the top-level algorithm for pruning multiple neurons in which the key step is to identify the multiple neurons to be considered:

I = MAB(* args)

Algorithm MAB 8.1 Algorithm for pruning multiple neurons or feature maps at one trail

INPUT: Time horizon T, Trained network, Input layers to be pruned.
OUTPUT: Pruned network
Let M is the number of neurons or feature maps that are played at one trail
Let I be the set of M indexes while i is a single index
Let X_I be the current reward of these neurons/feature maps
Let μ_I be the cumulative average reward of this arms/neurons/feature maps, initialized to zero
Let D be the random sample from training data set
Let N_I be set of M neurons or feature maps
Let $L(D|N)$ The loss function before pruning the M neurons or feature maps
Let $L(D|N')$ The loss function after pruning the M neurons or feature maps
Let Threshold be the loss in the performance that is allowed
Let K be the number of neurons in the chosen layer
Let n_i be the total current play time for neuron N_i
Let S_i be the number of successes and F_i the number of failures, both initialized to zero

1. for t=1 to T do /* start playing */
2. D = random example of the training data
3. Call the relevant MAB algorithm, returning the index of the selected neurons: $I = MAB(*arg)$
4. Perform forward propagation on D to compute $L(D|N)$
5. Get the output of N_I
6. Hold the value of the selected neurons of feature maps, $temp = N_I$
7. Set the neurons or feature maps to zero $N_I = 0$
8. Perform forward propagation on D to compute $L(D|N')$
9. Set the value of the neurons or feature maps to previous value, $N_I = temp$
10. $\delta L = L(D|N) - L(D|N')$
11. $X_{i,t}^g = REWARD(\delta L)$
12. Compute r_i for each arm such that $r_i = x_i * X_{i,t}^g$
13. Update the cumulative average reward of the current arms
$\mu_{I,t+1} = (n_I - 1)/n_I * \mu_{I,t} + 1/n_I * X_{I,t}^g$
14. end for
15. $PrunedModel = PrunedFunction(model, r)$
16. end main program

17. *Function* $PrunedFunction(model, rewards)$

18. Set to zero the weights that have most rewards
19. return PrunedModel
20. end Function

21. *Function* $REWARD(\delta L)$

22. if (bounded reward) then
23. $Reward += max(0, \delta L + Threshold)\ /Constant$
24. else /* Reward for Thompson Sampling */
25. if $\delta L < 0$ then reward=0, $F_{I,t+1} = F_{I,t} + 1$
26. else reward=1, $S_{I,t+1} = S_{I,t} + 1$
27. end if
28. end if
29. end Function

Figure 8.1: Pruning algorithm based on MP-MAB.

Two different multi-play multi-arm bandits are explored, namely one based on Thompson Sampling and the second based on UCB1, both of which are described below.

As described in Chapters 5, Thompson Sampling involves sampling the next arm to play based on drawing a random sample of the prior knowledge of the arms and then assign a reward. The main difference here, is that instead of selecting one neuron, the top *L* neurons are selected. Likewise, when adopting UCB1, the main difference is the selection of the top *L* neurons. The MAB functions for these two MAB based algorithms are presented in Figure 8.2 and Figure 8.3.

Algorithm 8.2 Multiple play Thompson Sampling for binary bandits for K arms

$Function\ \ MAB(t, S, F, K)$
Let $i \in \{1 \dots K\}$
Let A_t contain the sub of arms at time t
for i=1 to M do
Sample $\theta_i(t)$ for $Beta(S_i + 1, F_i + 1)$ distribution for all $i \in \{1 \dots K\}$
end for
Select A_I = top L arms ranked by $arg \max_{i \in \{1 \dots K\}} \theta_i(t)$
return I
end Function

Figure 8.2: MP-TS function where there are *K* neurons or feature maps.

Algorithm 8.3 Multiple play UCB1 for K arms

$Function\ \ MAB(t, \mu, K)$
Let $i \in \{1 \dots K\}$
Let A_t contain the sub of arms at time t
Select arm $A_{\boldsymbol{I}} = top\ L\ arms\ ranked\ by\ arg \max_{i \in \{1 \dots \boldsymbol{K}\}} (\mu_{i,t} + \sqrt{\frac{2lnt}{n_i}})$
return I
end Function

Figure 8.3: MP-UCB1 function where there are *K* neurons or feature maps.

8.3. Evaluation

This section presents the results of pruning multiple neurons and feature maps using the algorithms developed above. Both the proposed MAB algorithms were implemented using the Python programming language. All the reported experiments in this chapter were conducted using NVIDIA TITAN X.

Section 8.3.1 presents the results when the algorithms are used to prune neural networks trained on some UCI data sets and Section 8.3.2 presents the results when the algorithms are used to prune neural networks trained on deep learning data sets. Section 8.3.3 compares the performance of algorithms that prune multiple neurons (or feature maps) relative to those that prune a single neuron (or feature map) at a time. All the code used for these experiments is available online[26].

8.3.1. Testing on UCI Data sets

The proposed algorithms were tested on four widely used data sets from the UCI repository: Cancer, Iris, Valley, and Wine data sets. The methodology for training, pruning and testing the networks was explained in Section 6.2 and Table 6-5.

Figure 8.4 shows the accuracy of the models as the number of neurons (two neurons pruned at on play time) removed increases. The main conclusions that can be drawn from these results are:

- In general, pruning neurons on these data sets results in some improvement over the unpruned networks up to a point, after which performance decreases.
- Nearly 50% of the networks are prune until the performance sharply decreased.

8.3.2. Testing on Deep Learning Data sets

The proposed algorithms were tested on three widely used data sets from the deep learning benchmarks: Reuters, SVHN, and MNIST data sets. The methodology for training, pruning and testing the networks was explained in Section 6.2 and Table 6-6.

Table 8-1 shows the accuracy of the models as the number of neurons removed increases. The first column of the table shows the data set, the model, and the accuracy. The second column stand for the pruned layer or layers. The third column shows the number of neurons or feature maps that are pruned in one trail. The following two columns show the accuracy of the proposed algorithms. The final column shows the percentage of pruning relative to the original networks.

[26] https://github.com/SalemAmeen/banditsbook_Prune_many_one_play_based_on_loss_output

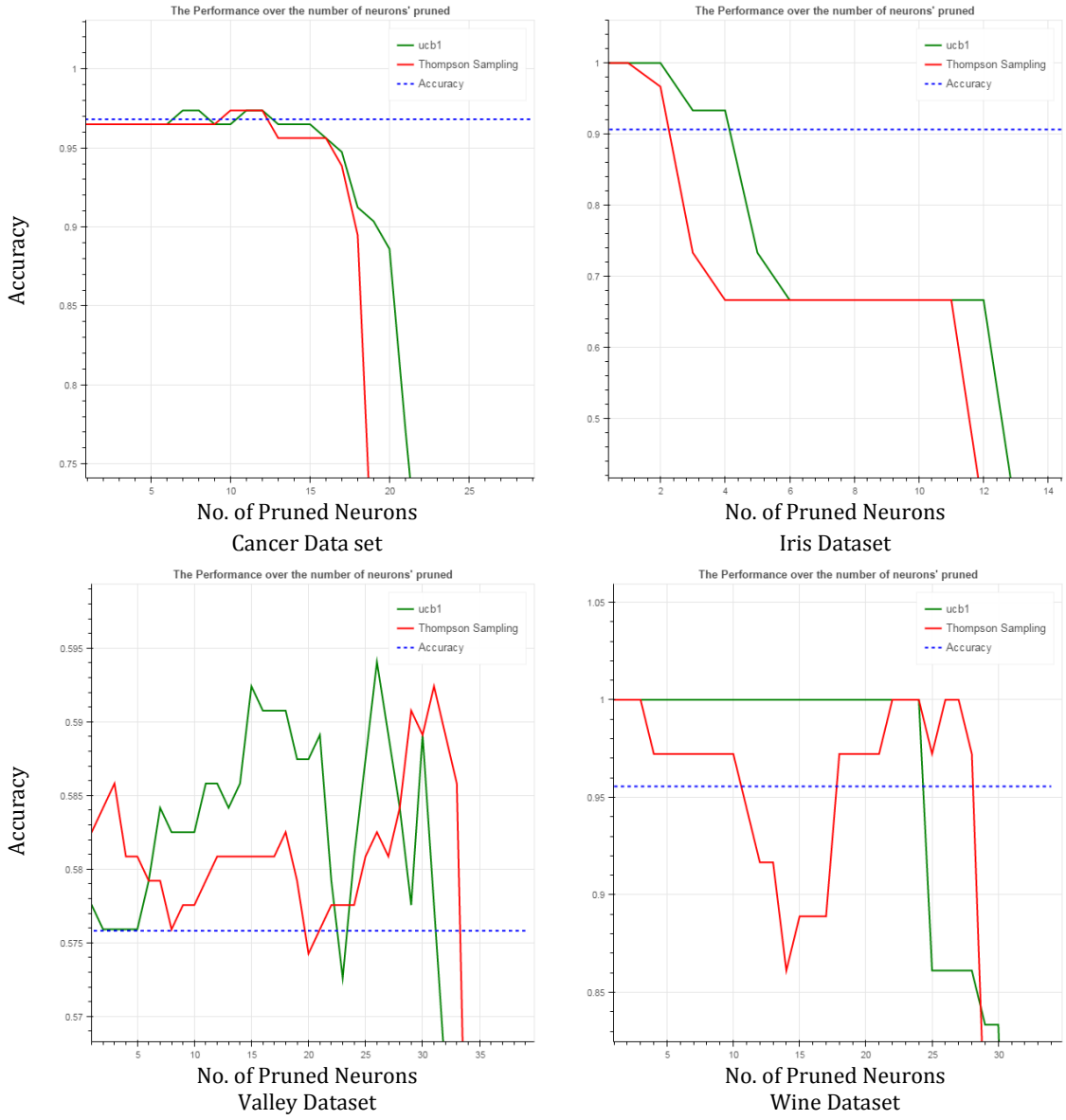


Figure 8.4: Pruning multiple neurons at one time.

The main conclusions that can be drawn from these results are:

- Pruning neurons and feature maps can be within a layer or across layers.
- In general, pruning neurons and feature maps on these data sets results in some improvement over the unpruned networks.
- Nearly 8-20% of the networks are pruned after which the performance decreases.
- Using two neurons that might cancel each other is more accurate when pruning the neurons or feature maps at the same layer. However, across the networks' layers using a higher number of feature maps or neurons leads to improvements in the performance

Model/Data set	Layer	# of neurons or feature maps	Model	MP-UCB1	MP-TS	Approxi. Pruned
MLP/Reuters	1st FC	2	↑ 0.79	↑ 0.79	↑ 0.79	17%
		5	↑ 0.79	↓ 0.78	↑ 0.79	17%
		10	↑ 0.79	↘ 0.76	↓ 0.75	17%
	2nd FC.	2	↑ 0.79	↓ 0.74	↓ 0.74	21%
		5	↑ 0.79	↓ 0.73	↓ 0.72	21%
		10	↑ 0.79	↘ 0.76	↓ 0.75	21%
	All FC.	2	↑ 0.79	↓ 0.74	↘ 0.75	22%
		5	↑ 0.79	↓ 0.74	↓ 0.74	22%
		10	↓ 0.79	↑ 0.8	↓ 0.79	22%
ConvNets/SVHN	1st Conv.	2	↓ 0.96	↑ 0.97	↓ 0.96	8%
		5	↑ 0.96	↓ 0.95	↑ 0.96	8%
		10	↑ 0.96	↓ 0.93	↓ 0.93	8%
	2nd Conv.	2	↑ 0.96	↑ 0.96	↑ 0.96	9%
		5	↑ 0.96	↘ 0.94	↓ 0.93	9%
		10	↑ 0.96	↓ 0.91	↓ 0.9	9%
	3rd Conv.	2	↑ 0.96	↑ 0.96	↑ 0.96	9%
		5	↑ 0.96	↓ 0.95	↑ 0.96	9%
		10	↑ 0.96	↓ 0.9	↓ 0.89	9%
	4th Conv.	2	↑ 0.96	↑ 0.96	↑ 0.96	11%
		5	↑ 0.96	↑ 0.96	↑ 0.96	11%
		10	↑ 0.96	→ 0.95	↓ 0.94	11%
	1st FC	2	↓ 0.96	↑ 0.97	↑ 0.97	17%
		5	↑ 0.96	↑ 0.96	↓ 0.95	17%
		10	↑ 0.96	↘ 0.93	↓ 0.92	17%
	All Conv.	2	↑ 0.96	↑ 0.96	↑ 0.96	12%
		5	↑ 0.96	↑ 0.96	↑ 0.96	12%
		10	↓ 0.96	↑ 0.98	→ 0.97	12%
LeNet/MNIST	1st Conv.	2	↑ 0.98	↑ 0.98	↑ 0.98	9%
		5	↑ 0.98	↓ 0.95	↘ 0.96	9%
		10	↑ 0.98	↓ 0.93	↓ 0.92	9%
	2nd Conv.	2	↑ 0.98	↑ 0.98	↑ 0.98	9%
		5	↑ 0.98	↓ 0.94	↓ 0.94	9%
		10	↑ 0.98	↓ 0.94	↓ 0.93	9%
	1st FC	2	↓ 0.98	↑ 0.99	↑ 0.99	28%
		5	↑ 0.98	↘ 0.96	↓ 0.95	28%
		10	↑ 0.98	↓ 0.93	↓ 0.93	28%
	All Conv.	2	↑ 0.98	↑ 0.98	↑ 0.98	11%
		5	↑ 0.98	↑ 0.98	↑ 0.98	11%
		10	↓ 0.98	↑ 0.99	↑ 0.99	11%

Table 8-1: Pruning using MP-TS and MP-UCB1. The green cells indicate that the method has good accuracy in contrast of red cell. The arrows point up if the error high, down if it is low or in right direction if it is in between.

8.3.3. Comparing with Pruning Single Neurons or Feature Maps

Finally, we compare the proposed algorithms with pruning single neurons and feature maps in one trail. The experiments use the same maximum number of trials and unpruned networks.

Table 8-2 shows the results, where the first column shows the data sets, the networks model, and the accuracy. The second column is the pruned layer which can be either a single layer or several layers. Then, each proposed algorithm has five columns. The first and second column in the proposed algorithm column shows whether neurons or feature maps are pruned. This is followed by the accuracy and the number of neurons or feature maps pruned. The last two columns show the percentage reduction in the size of the model as a result of pruning.

Model / Dataset / Accuracy		UCB1					Thompson Sampling					% single	% multiple
		Neuron	Feature Map	Multiple Prune			Neuron	Feature Map	Multiple Prune				
				2	5	10			2	5	10		
ConvNets LeNet Model On MNIST (0.98)	FC (layer 3)	⇧ 0.99		⇧ 0.99	⇨ 0.96	⇩ 0.93	⇧ 0.99		⇧ 0.99	↘ 0.95	⇩ 0.93	62	28
	Conv (layer 1)		⇧ 0.99	⇧ 0.98	⇨ 0.95	⇩ 0.93		⇧ 0.99	⇧ 0.98	⇨ 0.96	⇩ 0.92	13	9
	Conv (layer 2)		⇧ 0.98	⇧ 0.98	⇩ 0.94	⇩ 0.94		⇧ 0.98	⇧ 0.98	⇩ 0.94	⇩ 0.93	19	9
	Conv (All)		⇧ 0.99	⇩ 0.98	⇩ 0.98	⇧ 0.99		⇧ 0.99	⇩ 0.98	⇩ 0.98	⇧ 0.99	22	11
ConvNet on SVHN (0.96)	FC (layer 5)	⇧ 0.97		⇧ 0.97	⇧ 0.96	↘ 0.93	⇧ 0.97		⇧ 0.97	↗ 0.95	⇩ 0.92	39	17
	Conv (layer 1)		↗ 0.96	⇧ 0.97	⇨ 0.95	⇩ 0.93		↗ 0.96	↗ 0.96	↗ 0.96	⇩ 0.93	13	8
	Conv (layer 2)		⇧ 0.96	⇧ 0.96	↗ 0.94	⇩ 0.91		⇧ 0.96	⇧ 0.96	⇨ 0.93	⇩ 0.9	13	9
	Conv (layer 3)		⇧ 0.96	⇧ 0.96	⇧ 0.95	⇩ 0.9		⇧ 0.96	⇧ 0.96	⇧ 0.96	⇩ 0.89	16	9
	Conv (layer 4)		⇧ 0.96	⇧ 0.96	⇧ 0.96	⇨ 0.95		⇧ 0.96	⇧ 0.96	⇧ 0.96	⇩ 0.94	22	11
	Conv (All layers)		⇨ 0.97	⇩ 0.96	⇩ 0.96	⇧ 0.98		⇩ 0.96	⇩ 0.96	⇩ 0.96	⇨ 0.97	27	12
MLP on Reuters (0.79)	FC (layer 2)	⇧ 0.79		↘ 0.74	⇩ 0.73	⇨ 0.76	⇧ 0.79		↘ 0.74	⇩ 0.72	⇨ 0.75	50	21

Table 8-2: Summary of results based on accuracy from three common data sets that were used in pruning based on single or multiple neurons or feature maps. Cells shaded black indicate there is no result. Cells shaded green has best accuracy while red has the worse.

The main conclusion that can be drawn from these results is that in general, pruning single neurons and feature maps results in a greater proportion of neurons being pruned than pruning multiple neurons. This was a surprising outcome given our expectations that pruning multiple neurons would detect neurons that cancel each other as suggested in the work of Hinton et al. [103] and Reed [111]. The experiments here are limited to seven data sets, and

further work in this area is needed to fully understand why the effects expected have not materialised. One possible direction of work could be to review the measure used in Equation 8.1.

8.4. Summary

This chapter was motivated by a view that removing multiple neurons at the same time would lead to detection of correlated neurons that were unnecessary or even had a negative effect on the accuracy of a network (e.g. Reed [111], Hinton et al. [103]). Hence, this chapter developed two pruning algorithms that prune multiple neurons and feature maps at the same time.

The empirical evaluation on seven data sets shows that pruning multiple neurons can reduce the size of the models without affecting the accuracy of the model. However, contrary to our expectations (based on the view of other authors [111]), an empirical comparison also showed that pruning single neurons or feature maps at a time results in greater pruning. Further work and experimentation is therefore needed to understand why pruning multiple neurons using MAB methods was not as effective as pruning single neurons at a time.

9. Conclusion and Future Work

9.1. Introduction

Pruning mechanisms are an important part of practical learning algorithms for deep neural networks [134, 159, 219, 220]. This thesis explore the use of multi-armed bandits for improving the process of pruning neural networks. This chapter offers a summary of the research work described in this thesis, the main findings, and some directions for future work. The rest of this chapter is organised as follows: Section 9.2 gives a summary of the work presented. Section 9.3 shows the main findings in the context of the research questions identified in Chapter 1. Finally, Section 9.4 recommends directions for future work.

9.2. Summary

The first stages of the study involved surveying the literature on deep learning and pruning methods. This identified a range of pruning methods including OBD, OBS, Network Pruning, Local Bottlenecks, Skeletonization, Iterative Pruning, Channel Level Acceleration, Network Trimming, Entropy Based Pruning and Pruning Smallest Filters which are described in Chapter 3. Although OBD and OBS have good theoretical foundations, they require the inversion of a Hessian matrix that makes these methods computationally expensive. Chapter 4 surveyed multi-arm bandit (MAB) methods including UCB based methods, Epsilon-Greedy, Win-Stay; Lose-Shift (WSLS), Softmax, Hedge, EXP3 and Thompson Sampling.

Deep learning neural networks are organised in layers of neurons that are connected by weights. Pruning can therefore occur at several layers of granularity: individual weights can be pruned, neurons can be pruned, feature maps can be pruned, and multiple neurons and feature maps can also be pruned. The study explored the use of MABs for each of these possibilities:

- Chapter 5 presented six new algorithms for pruning weights based on: Epsilon-Greedy, Win-Stay; Lose-Shift, UCB1, KL-UCB, Thompson Sampling and BayesUCB. An empirical evaluation of these algorithms indicated that:
    - The MAB pruning algorithms produced better results than the original models.
    - Some of the proposed MAB methods outperformed the other pruning methods.
    - The proposed methods had manageable time to prune the weights of a network in contrast to other methods like OBS and OBD.
- Chapter 6 presented algorithms that remove neurons, thereby removing all the weights that are associated with the neuron that is removed. Multiple pruning algorithms based on MAB are introduced. An empirical evaluation of these algorithms shows that:
    - UCB1 and Thompson Sampling pruning show the best results among the proposed algorithms.
    - Adversarial bandits are not as effective as the other MAB pruning algorithms.
- Chapter 7 discussed the development of pruning algorithms based on Thompson Sampling and UCB1 to prune feature maps with a view to speeding up ConvNets. An empirical evaluation of these algorithms, presented in Table 7-2, shows that:
    - MAB based pruning can be an effective way of pruning feature maps and can reduce the number feature maps significantly. For example, in the LeNet model trained on the MNIST data, there was a reduction of over 21% in the number of feature maps
    - The use of UCB1 and Thompson Sampling for pruning feature maps produced results that were ranked higher than both the greedy and magnitude based approaches to pruning feature maps.
    - Applying MAB based on all the layers produced better results than applying them layer by layer.

- Chapter 8 extends two MAB algorithms to play with many arms instead of one which has the potential to remove neurons and feature maps that cancel each other. However, the results of an empirical evaluation are not as good as the other MAB methods. One likely reason for this is that the training process adopted may have already led to minimisation of neurons that interact in a negative manner.

A statistical analysis of the outcomes was first conducted using the Friedman test and then followed up with the Nemenyi post-hoc test for comparing different methods. Interesting outcomes were obtained and these are discussed more in the following section.

9.3. Contribution and Main Findings

This section presents the main findings from the research work described in the thesis. As initially presented in Chapter 1, the three main research questions to be addressed in this study were:

a) How well do existing algorithms for pruning neural networks perform?

b) Can MAB algorithms be developed for pruning and which MAB methods work best?

c) How does the performance of the MAB based pruning methods compare with other methods?

The thesis aimed to answer these questions, and the main findings are summarised below.

a) How well do existing algorithms for pruning neural networks perform?

The study carried out a comprehensive review of the literature, implemented a selection of the algorithms and applied them on several data sets. The main conclusions with respect to this question are:

- The performance of existing techniques is proportional to the size and depth of deep neural networks. As the networks becomes bigger and deeper, the pruning techniques perform better, unless these algorithms have very slow computation time (e.g., OBD and OBS) in which case they cannot be used in practice [132].
- Most of the existing pruning algorithms need to retrain the pruned network after pruning to maintain the performance of the network [36, 114].

b) Can MAB algorithms be developed for pruning and which methods work best?

- Different MAB pruning algorithms were developed to prune weights, neurons, and feature maps. The methods developed and implemented using TensorFlow and Python include UCB1, KL-UCB, BayesUCB, WSLS, Softmax, Hedge, EXP3 and Thompson Sampling.
- The algorithms were evaluated on 23 data sets from the UCI and Kaggle repositories, and using eight deep learning data sets implemented using ConvNets, MLP and RNNs.
- The results show that in general, pruning based on UCB1 and Thompson Sampling showed the best performance among the MAB pruning algorithms. The evaluation also showed that the WSLS does not perform well.

c) How does the performance of the MAB based pruning methods compare with other methods?

- The MAB based algorithms were compared with several existing methods, including OBD, OBS, Network pruning, random pruning, magnitude based pruning and pruning based on activation.
- As expected, OBS was very slow given the need to invert a Hessian matrix. For example, it was 512 times slower than the UCB1 on the SPECTF Heart data set.
- Pruning based on MAB presented the best results for pruning the weights, neurons, and feature maps.

9.4. Future Work

The research presented in this thesis has shown promising results, and there are a number of directions for future work, such as:

- The literature includes some uses of UCB1 based on bounded rewards that have resulted in good performance over the other proposed algorithms [221, 222]. In addition, Thompson Sampling, which uses binary rewards, produced the second-best results. Therefore, using Thompson Sampling with bounded rewards might give better results over binary rewards and would be worth exploring in the future.

- Instead of computing the saliency based on forward propagation [113, 114, 119-121, 140], back propagation can be used to compute the change of the gradient for particular neuron of feature [115]
- One of the biggest successes in deep learning is using GPUs and parallel computing to train deep learning. It is therefore worth implementing the MAB pruning algorithms to take advantage of parallel computing to speed up the process of pruning [223, 224].
- The thesis also considered the use of multi-play bandits for identifying several neurons that can be deleted. Apart from OBS, and as far as the author is aware, there is no other method that aims to remove multiple neurons. The evaluations show that selecting multiple neurons does not necessarily produce better results than selecting single neurons which was a little surprising given that the literature suggests that there are problems with neurons cancelling each other out [103, 111]. This unexpected outcome might be because the training process employed avoided such neurons, but needs further exploration.

10. References

[1] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature,* vol. 323, pp. 533-536, 1986.

[2] G. P. Zhang, "Neural networks for classification: a survey," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews),* vol. 30, pp. 451-462, 2000.

[3] A. Vellido, P. J. Lisboa, and J. Vaughan, "Neural networks in business: a survey of applications (1992–1998)," *Expert Systems with applications,* vol. 17, pp. 51-70, 1999.

[4] Y. Le Cun, L. Bottou, and Y. Bengio, "Reading checks with multilayer graph transformer networks," in *Acoustics, Speech, and Signal Processing, 1997. ICASSP-97., 1997 IEEE International Conference on*, 1997, pp. 151-154.

[5] B. Šter and A. Dobnikar, "Neural networks in medical diagnosis: Comparison with other methods," in *International Conference on Engineering Applications of Neural Networks*, 1996, pp. 427-30.

[6] F. Amato, A. López, E. M. Peña-Méndez, P. Vaňhara, A. Hampl, and J. Havel, "Artificial neural networks in medical diagnosis," ed: Elsevier, 2013.

[7] J. Khan, J. S. Wei, M. Ringner, L. H. Saal, M. Ladanyi, F. Westermann, *et al.*, "Classification and diagnostic prediction of cancers using gene expression profiling and artificial neural networks," *Nature medicine,* vol. 7, p. 673, 2001.

[8] W. G. Baxt, "Application of artificial neural networks to clinical medicine," *The lancet,* vol. 346, pp. 1135-1138, 1995.

[9] W. T. Miller, P. J. Werbos, and R. S. Sutton, *Neural networks for control*: MIT press, 1995.

[10] D. Pham and X. Liu, "NEURAL NETIWORKS FOR IDENTIFICATION, PREDICTION AND CONTROL," 1995.

[11] F. Lewis, S. Jagannathan, and A. Yesildirak, *Neural network control of robot manipulators and non-linear systems*: CRC Press, 1998.

[12] G. Hinton, S. Osindero, and Y. Teh, "A Fast Learning Algorithm for Deep Belief Nets," *Neural Computation,* vol. 18, pp. 1527-1554, 2006.

[13] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science,* vol. 313, pp. 504-507, 2006.

[14] H. Pratt, F. Coenen, D. M. Broadbent, S. P. Harding, and Y. Zheng, "Convolutional neural networks for diabetic retinopathy," *Procedia Computer Science,* vol. 90, pp. 200-205, 2016.

[15] R. Qian, Y. Yue, F. Coenen, and B. Zhang, "Visual attribute classification using feature selection and convolutional neural network," in *Signal Processing (ICSP), 2016 IEEE 13th International Conference on*, 2016, pp. 649-653.

[16] Y. Xia, B. Zhang, and F. Coenen, "Face Occlusion Detection Using Deep Convolutional Neural Networks," *International Journal of Pattern Recognition and Artificial Intelligence,* vol. 30, p. 1660010, 2016.

[17] S. Ameen and S. Vadera, "A convolutional neural network to classify American Sign Language fingerspelling from depth and colour images," *Expert Systems,* 2017.

[18] R. Qian, Y. Yue, F. Coenen, and B. Zhang, "Traffic sign recognition with convolutional neural network based on max pooling positions," in *Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNC-FSKD), 2016 12th International Conference on*, 2016, pp. 578-582.

[19] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, "How transferable are features in deep neural networks?," in *Advances in Neural Information Processing Systems*, 2014, pp. 3320-3328.

[20] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature,* vol. 521, pp. 436-444, 05/28/print 2015.

[21] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and F. Li, "Imagenet: A large-scale hierarchical image database," in *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, 2009, pp. 248-255.

[22] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097-1105.

[23] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[24] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, *et al.*, "Going deeper with convolutions," *arXiv preprint arXiv:1409.4842*, 2014.

[25] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," *arXiv preprint arXiv:1512.03385*, 2015.

[26] G. Huang, Z. Liu, K. Q. Weinberger, and L. van der Maaten, "Densely connected convolutional networks," *arXiv preprint arXiv:1608.06993*, 2016.

[27] K. Gurney, *An introduction to neural networks*: CRC press, 1997.

[28] C. A. Walsh, "Peter Huttenlocher (1931-2013)," *Nature*, vol. 502, pp. 172-172, 2013.

[29] M. Denil, B. Shakibi, L. Dinh, and N. de Freitas, "Predicting parameters in deep learning," in *Advances in Neural Information Processing Systems*, 2013, pp. 2148-2156.

[30] V. Sze, T.-J. Yang, and Y.-H. Chen, "Designing Energy-Efficient Convolutional Neural Networks using Energy-Aware Pruning," 2017.

[31] N. D. Lane and P. Georgiev, "Can deep learning revolutionize mobile sensing?," in *Proceedings of the 16th International Workshop on Mobile Computing Systems and Applications*, 2015, pp. 117-122.

[32] M. A. Alsheikh, D. Niyato, S. Lin, H.-P. Tan, and Z. Han, "Mobile big data analytics using deep learning and apache spark," *IEEE network*, vol. 30, pp. 22-29, 2016.

[33] N. D. Lane, S. Bhattacharya, P. Georgiev, C. Forlivesi, L. Jiao, L. Qendro, *et al.*, "Deepx: A software accelerator for low-power deep learning inference on mobile devices," in *Information Processing in Sensor Networks (IPSN), 2016 15th ACM/IEEE International Conference on*, 2016, pp. 1-12.

[34] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, *et al.*, "EIE: efficient inference engine on compressed deep neural network," *arXiv preprint arXiv:1602.01528*, 2016.

[35] A. Angelova, A. Krizhevsky, V. Vanhoucke, A. S. Ogale, and D. Ferguson, "Real-Time Pedestrian Detection with Deep Network Cascades," in *BMVC*, 2015, p. 4.

[36] M. C. Mozer and P. Smolensky, "Skeletonization: A Technique for Trimming the Fat from a Network via Relevance Assessment; CU-CS-421-89," 1989.

[37] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. K. Warmuth, "Occam's razor," *Readings in machine learning*, pp. 201-204, 1990.

[38] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The bulletin of mathematical biophysics,* vol. 5, pp. 115-133, 1943.

[39] A. Zhou, A. Yao, Y. Guo, L. Xu, and Y. Chen, "Incremental network quantization: Towards lossless cnns with low-precision weights," *arXiv preprint arXiv:1702.03044,* 2017.

[40] R. Collobert, K. Kavukcuoglu, and C. Farabet, "Implementing neural networks efficiently," in *Neural Networks: Tricks of the Trade*, ed: Springer, 2012, pp. 537-557.

[41] W. R. Thompson, "On the likelihood that one unknown probability exceeds another in view of the evidence of two samples," *Biometrika,* vol. 25, pp. 285-294, 1933.

[42] W. R. Thompson, "On the theory of apportionment," *American Journal of Mathematics,* vol. 57, pp. 450-456, 1935.

[43] H. Robbins, "Some aspects of the sequential design of experiments," *Bulletin of the American Mathematical Society,* vol. 58, pp. 527-535, 1952.

[44] M. Babaioff, Y. Sharma, and A. Slivkins, "Characterizing truthful multi-armed bandit mechanisms," in *Proceedings of the 10th ACM conference on Electronic commerce*, 2009, pp. 79-88.

[45] N. R. Devanur and S. M. Kakade, "The price of truthfulness for pay-per-click auctions," in *Proceedings of the 10th ACM conference on Electronic commerce*, 2009, pp. 99-106.

[46] S. N. Durlauf and L. Blume, *The new Palgrave dictionary of economics* vol. 6: Palgrave Macmillan Basingstoke, 2008.

[47] D. Lamberton, G. Pagès, and P. Tarrès, "When can the two-armed bandit algorithm be trusted?," *Annals of Applied Probability,* pp. 1424-1454, 2004.

[48] S. Gelly and Y. Wang, "Exploration exploitation in go: UCT for Monte-Carlo go," in *NIPS: Neural Information Processing Systems Conference On-line trading of Exploration and Exploitation Workshop,* 2006.

[49] R. D. Kleinberg, "Nearly tight bounds for the continuum-armed bandit problem," in *Advances in Neural Information Processing Systems*, 2004, pp. 697-704.

[50] P.-A. Coquelin and R. Munos, "Bandit algorithms for tree search," *arXiv preprint cs/0703062,* 2007.

[51] R. Kleinberg, A. Slivkins, and E. Upfal, "Multi-armed bandits in metric spaces," in *Proceedings of the fortieth annual ACM symposium on Theory of computing*, 2008, pp. 681-690.

[52] S. Bubeck, G. Stoltz, C. Szepesvári, and R. Munos, "Online optimization in X-armed bandits," in *Advances in Neural Information Processing Systems*, 2009, pp. 201-208.

[53] V. Mnih, C. Szepesvári, and J.-Y. Audibert, "Empirical bernstein stopping," in *Proceedings of the 25th international conference on Machine learning*, 2008, pp. 672-679.

[54] R. Busa-Fekete and B. Kégl, "Fast boosting using adversarial bandits," in *27th International Conference on Machine Learning (ICML 2010)*, 2010, pp. 143-150.

[55] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction* vol. 1: MIT press Cambridge, 1998.

[56] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, *et al.*, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[57] C. R. Kothari, *Research methodology: Methods and techniques*: New Age International, 2004.

[58] T. M. Mitchell, *The discipline of machine learning* vol. 3: Carnegie Mellon University, School of Computer Science, Machine Learning Department, 2006.

[59] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*: MIT press, 2016.

[60] A. Karpathy, "Connecting Images and Natural Language," Stanford University, 2016.

[61] C. M. Bishop, *Pattern recognition and machine learning*: springer, 2006.

[62] Y. S. Abu-Mostafa, M. Magdon-Ismail, and H.-T. Lin, *Learning from data* vol. 4: AMLBook New York, NY, USA:, 2012.

[63] Y. Chauvin, "A back-propagation algorithm with optimal use of hidden units," in *Advances in neural information processing systems*, 1989, pp. 519-526.

[64] D. Weigend, "Back-propagation, weight-elimination and time series prediction," in *Proceedings 1990 Connectionist Models Summer School*, 1990, pp. 105-116.

[65] A. S. Weigend, D. E. Rumelhart, and B. A. Huberman, "Generalization by weight-elimination applied to currency exchange rate prediction," in *Neural Networks, 1991., IJCNN-91-Seattle International Joint Conference on*, 1991, pp. 837-841.

[66] A. S. Weigend, D. E. Rumelhart, and B. A. Huberman, "Generalization by weight-elimination with application to forecasting," in *Advances in neural information processing systems*, 1991, pp. 875-882.

[67] C. Ji, R. R. Snapp, and D. Psaltis, "Generalizing smoothness constraints from discrete samples," *Neural Computation*, vol. 2, pp. 188-197, 1990.

[68] M. Avriel, *Nonlinear programming: analysis and methods*: Courier Corporation, 2003.

[69] A. Cauchy, "Méthode générale pour la résolution des systemes d'équations simultanées," *Comp. Rend. Sci. Paris,* vol. 25, pp. 536-538, 1847.

[70] M. Li, T. Zhang, Y. Chen, and A. J. Smola, "Efficient mini-batch training for stochastic optimization," in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2014, pp. 661-670.

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

[72] R. S. Sutton, "Two problems with backpropagation and other steepest-descent learning procedures for networks," in *Proc. 8th annual conf. cognitive science society*, 1986, pp. 823-831.

[73] 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.

[74] J. Dean, G. S. Corrado, R. Monga, K. Chen, M. Devin, Q. V. Le, *et al.*, "Large Scale Distributed Deep Networks," *Google Inc,* 2012.

[75] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *EMNLP*, 2014, pp. 1532-1543.

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

[77] 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.

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

[79] A. E. Bryson, W. F. Denham, and S. E. Dreyfus, "Optimal programming problems with inequality constraints," *AIAA journal,* vol. 1, pp. 2544-2550, 1963.

[80] A. G. Baydin, B. A. Pearlmutter, and A. A. Radul, "Automatic differentiation in machine learning: a survey," *arXiv preprint arXiv:1502.05767,* 2015.

[81] Werbos, "Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Science," PhD thesis, Harvard University, 1974.

[82] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain," *Psychological review,* vol. 65, p. 386, 1958.

[83] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, pp. 2278-2324, 1998.

[84] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *Computer Vision–ECCV 2014*, ed: Springer, 2014, pp. 818-833.

[85] M. D. Zeiler, D. Krishnan, G. W. Taylor, and R. Fergus, "Deconvolutional networks," in *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, 2010, pp. 2528-2535.

[86] T. D. Kulkarni, W. F. Whitney, P. Kohli, and J. Tenenbaum, "Deep convolutional inverse graphics network," in *Advances in Neural Information Processing Systems*, 2015, pp. 2539-2547.

[87] J. Redmon and A. Farhadi, "YOLO9000: better, faster, stronger," *arXiv preprint arXiv:1612.08242*, 2016.

[88] J. L. Elman, "Finding structure in time," *Cognitive science*, vol. 14, pp. 179-211, 1990.

[89] T. Kohonen, "Self-organized formation of topologically correct feature maps," *Biological cybernetics*, vol. 43, pp. 59-69, 1982.

[90] G. E. Hinton and T. J. Sejnowski, "Learning and releaming in Boltzmann machines," *Parallel Distrilmted Processing*, vol. 1, 1986.

[91] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle, "Greedy layer-wise training of deep networks," *Advances in neural information processing systems*, vol. 19, p. 153, 2007.

[92] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, *et al.*, "Generative adversarial nets," in *Advances in Neural Information Processing Systems*, 2014, pp. 2672-2680.

[93] H. Bourlard and Y. Kamp, "Auto-association by multilayer perceptrons and singular value decomposition," *Biological Cybernetics*, vol. 59, pp. 291-294, 1988/09/01 1988.

[94] C. P. Marc'Aurelio Ranzato, S. Chopra, and Y. LeCun, "Efficient learning of sparse representations with an energy-based model," in *Proceedings of NIPS*, 2007.

[95] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 1026-1034.

[96] I. J. Goodfellow, D. Warde-Farley, M. Mirza, A. C. Courville, and Y. Bengio, "Maxout networks," *ICML (3)*, vol. 28, pp. 1319-1327, 2013.

[97] I. Sutskever, J. Martens, and G. E. Hinton, "Generating text with recurrent neural networks," in *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, 2011, pp. 1017-1024.

[98] Y. Wu, S. Zhang, Y. Zhang, Y. Bengio, and R. R. Salakhutdinov, "On multiplicative integration with recurrent neural networks," in *Advances In Neural Information Processing Systems*, 2016, pp. 2856-2864.

[99] B. A. Plummer, L. Wang, C. M. Cervantes, J. C. Caicedo, J. Hockenmaier, and S. Lazebnik, "Flickr30k entities: Collecting region-to-phrase correspondences for richer image-to-sentence models," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 2641-2649.

[100] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, pp. 1735-1780, 1997.

[101] S. Sukhbaatar, A. Szlam, J. Weston, and R. Fergus, "End-To-End Memory Networks," in *Advances in Neural Information Processing Systems*, 2015, pp. 2431-2439.

[102] H. Larochelle, Y. Bengio, m. Louradour, and P. Lamblin, "Exploring Strategies for Training Deep Neural Networks," *J. Mach. Learn. Res.*, vol. 10, pp. 1-40, 2009.

[103] 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.

[104] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, *et al.*, "Imagenet large scale visual recognition challenge," *arXiv preprint arXiv:1409.0575*, 2014.

[105] L. Wan, M. Zeiler, S. Zhang, Y. L. Cun, and R. Fergus, "Regularization of neural networks using dropconnect," in *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, 2013, pp. 1058-1066.

[106] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, 2015.

[107] G. F. Montufar, R. Pascanu, K. Cho, and Y. Bengio, "On the number of linear regions of deep neural networks," in *Advances in neural information processing systems*, 2014, pp. 2924-2932.

[108] C. Szegedy, S. Ioffe, and V. Vanhoucke, "Inception-v4, inception-resnet and the impact of residual connections on learning," *arXiv preprint arXiv:1602.07261*, 2016.

[109] G. Huang, Y. Sun, Z. Liu, D. Sedra, and K. Q. Weinberger, "Deep networks with stochastic depth," in *European Conference on Computer Vision*, 2016, pp. 646-661.

[110] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, *et al.*, "Caffe: Convolutional architecture for fast feature embedding," in *Proceedings of the ACM International Conference on Multimedia*, 2014, pp. 675-678.

[111] R. Reed, "Pruning algorithms-a survey," *Neural Networks, IEEE Transactions on*, vol. 4, pp. 740-747, 1993.

[112] M. G. Augasta and T. Kathirvalavakumar, "Pruning algorithms of neural networks—a comparative study," *Central European Journal of Computer Science*, vol. 3, pp. 105-115, 2013.

[113] J. Kruschke, "Creating local and distributed bottlenecks in hidden layers of back-propagation networks," *Proceedings 1998 Connectionist Models Summer School*, pp. 120-126, 1988.

[114] N. Wolfe, A. Sharma, L. Drude, and B. Raj, "The Incredible Shrinking Neural Network: New Perspectives on Learning Representations Through The Lens of Pruning," *arXiv preprint arXiv:1701.04465*, 2017.

[115] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz, "Pruning Convolutional Neural Networks for Resource Efficient Transfer Learning," *arXiv preprint arXiv:1611.06440*, 2016.

[116] S. J. Hanson and L. Y. Pratt, "Comparing biases for minimal network construction with back-propagation," in *Advances in neural information processing systems*, 1989, pp. 177-185.

[117] J. Hertz, A. Krogh, and R. G. Palmer, *Introduction to the theory of neural computation* vol. 1: Basic Books, 1991.

[118] M. A. Arbib, "The handbook of brain theory and neural networks," 1995.

[119] A. Polyak and L. Wolf, "Channel-level acceleration of deep face representations," *IEEE Access*, vol. 3, pp. 2163-2175, 2015.

[120] H. Hu, R. Peng, Y.-W. Tai, and C.-K. Tang, "Network Trimming: A data-driven neuron pruning approach towards efficient deep architectures," *arXiv preprint arXiv:1607.03250*, 2016.

[121] J.-H. Luo and J. Wu, "An Entropy-based Pruning Method for CNN Compression," *arXiv preprint arXiv:1706.05791*, 2017.

[122] N. Nikolaev and H. Iba, *Adaptive learning of polynomial networks: genetic programming, backpropagation and Bayesian methods*: Springer Science & Business Media, 2006.

[123] W. Finnoff, F. Hergert, and H. G. Zimmermann, "Improving model selection by nonconvergent methods," *Neural Networks,* vol. 6, pp. 771-783, 1993.

[124] R. Neuneier and H. Zimmermann, "How to train neural networks," *Neural networks: tricks of the trade,* pp. 550-550, 1998.

[125] Y. LeCun, J. S. Denker, S. A. Solla, R. E. Howard, and L. D. Jackel, "Optimal brain damage," in *NIPs*, 1989.

[126] B. Hassibi, D. G. Stork, and G. J. Wolff, "Optimal brain surgeon and general network pruning," in *Neural Networks, 1993., IEEE International Conference on*, 1993, pp. 293-299.

[127] B. Hassibi, D. G. Stork, G. Wolff, and T. Watanabe, "Optimal Brain Surgeon: Extensions and performance comparison," 1994.

[128] B. Hassibi and D. G. Stork, *Second order derivatives for network pruning: Optimal brain surgeon*: Morgan Kaufmann, 1993.

[129] M. D. Collins and P. Kohli, "Memory bounded deep convolutional networks," *arXiv preprint arXiv:1412.1442,* 2014.

[130] G. James, D. Witten, T. Hastie, and R. Tibshirani, *An introduction to statistical learning*: Springer, 2013.

[131] M. Gupta, L. Jin, and N. Homma, *Static and dynamic neural networks: from fundamentals to advanced theory*: John Wiley & Sons, 2004.

[132] S. Srinivas and R. V. Babu, "Data-free parameter pruning for Deep Neural Networks," *arXiv preprint arXiv:1507.06149,* 2015.

[133] A. U. Levin, T. K. Leen, and J. E. Moody, "Fast pruning using principal components," in *Advances in neural information processing systems*, 1994, pp. 35-42.

[134] S. Han, J. Pool, J. Tran, and W. J. Dally, "Learning both weights and connections for efficient neural networks," *arXiv preprint arXiv:1506.02626,* 2015.

[135] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *The Journal of Machine Learning Research,* vol. 15, pp. 1929-1958, 2014.

[136] A. See, M.-T. Luong, and C. D. Manning, "Compression of Neural Machine Translation Models via Pruning," *arXiv preprint arXiv:1606.09274,* 2016.

[137] S. Narang, G. Diamos, S. Sengupta, and E. Elsen, "Exploring Sparsity in Recurrent Neural Networks," *arXiv preprint arXiv:1704.05119,* 2017.

[138] S. Han, H. Mao, and W. J. Dally, "Deep Compression: Compressing Deep Neural Network with Pruning, Trained Quantization and Huffman Coding," 2015.

[139] S. Han, J. Kang, H. Mao, Y. Hu, X. Li, Y. Li, *et al.*, "ESE: Efficient Speech Recognition Engine with Sparse LSTM on FPGA," in *FPGA*, 2017, pp. 75-84.

[140] G. Castellano, A. M. Fanelli, and M. Pelillo, "An iterative pruning algorithm for feedforward neural networks," *IEEE transactions on Neural networks*, vol. 8, pp. 519-531, 1997.

[141] Å. Björck and T. Elfving, "Accelerated projection methods for computing pseudoinverse solutions of systems of linear equations," *BIT Numerical Mathematics*, vol. 19, pp. 145-163, 1979.

[142] G. Castellano and A. M. Fanelli, "Variable selection using neural-network models," *Neurocomputing*, vol. 31, pp. 1-13, 2000.

[143] A. Fangju, "A new pruning algorithm for Feedforward Neural Networks," in *Advanced Computational Intelligence (IWACI), 2011 Fourth International Workshop on*, 2011, pp. 286-289.

[144] A. Ben-Israel and T. N. Greville, *Generalized inverses: theory and applications* vol. 15: Springer Science & Business Media, 2003.

[145] B. Han, Z. Zhang, C. Xu, B. Wang, G. Hu, L. Bai, *et al.*, "Deep Face Model Compression Using Entropy-based Filter Selection," in *Proceedings ICIAP 2017: Lecture Notes in Computer Science*, 2017.

[146] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning Filters for Efficient ConvNets," *arXiv preprint arXiv:1608.08710*, 2016.

[147] W. Chen, J. T. Wilson, S. Tyree, K. Q. Weinberger, and Y. Chen, "Compressing Neural Networks with the Hashing Trick," *arXiv preprint arXiv:1504.04788*, 2015.

[148] W. Chen, J. T. Wilson, S. Tyree, K. Q. Weinberger, and Y. Chen, "Compressing Convolutional Neural Networks," *arXiv preprint arXiv:1506.04449*, 2015.

[149] T. N. Sainath, B. Kingsbury, V. Sindhwani, E. Arisoy, and B. Ramabhadran, "Low-rank matrix factorization for deep neural network training with high-dimensional output targets," in *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, 2013, pp. 6655-6659.

[150] J. Xue, J. Li, and Y. Gong, "Restructuring of deep neural network acoustic models with singular value decomposition," in *INTERSPEECH*, 2013, pp. 2365-2369.

[151] Y. Gong, L. Liu, M. Yang, and L. Bourdev, "Compressing Deep Convolutional Networks using Vector Quantization," *arXiv preprint arXiv:1412.6115*, 2014.

[152] E. L. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus, "Exploiting linear structure within convolutional networks for efficient evaluation," in *Advances in Neural Information Processing Systems*, 2014, pp. 1269-1277.

[153] F. N. Iandola, M. W. Moskewicz, K. Ashraf, S. Han, W. J. Dally, and K. Keutzer, "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and< 1MB model size," *arXiv preprint arXiv:1602.07360*, 2016.

[154] M. Jaderberg, A. Vedaldi, and A. Zisserman, "Speeding up convolutional neural networks with low rank expansions," *arXiv preprint arXiv:1405.3866*, 2014.

[155] X. Zhang, J. Zou, X. Ming, K. He, and J. Sun, "Efficient and accurate approximations of nonlinear convolutional networks," *arXiv preprint arXiv:1411.4229*, 2014.

[156] H. Li, R. Zhao, and X. Wang, "Highly efficient forward and backward propagation of convolutional neural networks for pixelwise classification," *arXiv preprint arXiv:1412.4526*, 2014.

[157] V. Lebedev, Y. Ganin, M. Rakhuba, I. Oseledets, and V. Lempitsky, "Speeding-up convolutional neural networks using fine-tuned cp-decomposition," *arXiv preprint arXiv:1412.6553*, 2014.

[158] P. Vincent, A. de Brébisson, and X. Bouthillier, "Efficient exact gradient update for training deep networks with very large sparse targets," in *Advances in Neural Information Processing Systems*, 2015, pp. 1108-1116.

[159] M. Mathieu, M. Henaff, and Y. LeCun, "Fast training of convolutional networks through ffts," *arXiv preprint arXiv:1312.5851*, 2013.

[160] T. Highlander and A. Rodriguez, "Very Efficient Training of Convolutional Neural Networks using Fast Fourier Transform and Overlap-and-Add," *arXiv preprint arXiv:1601.06815*, 2016.

[161] O. Rippel, J. Snoek, and R. P. Adams, "Spectral representations for convolutional neural networks," in *Advances in Neural Information Processing Systems*, 2015, pp. 2449-2457.

[162] H. Pratt, B. Williams, F. Coenen, and Y. Zheng, "FCNN: Fourier Convolutional Neural Networks."

[163] K. J. Arrow, D. Blackwell, and M. A. Girshick, "Bayes and minimax solutions of sequential decision problems," *Econometrica, Journal of the Econometric Society*, pp. 213-244, 1949.

[164] T. L. Lai and H. Robbins, "Asymptotically efficient adaptive allocation rules," *Advances in applied mathematics*, vol. 6, pp. 4-22, 1985.

[165] G. Burtini, J. Loeppky, and R. Lawrence, "A Survey of Online Experiment Design with the Stochastic Multi-Armed Bandit," *arXiv preprint arXiv:1510.00757,* 2015.

[166] L. Zhou, "A survey on contextual multi-armed bandits," *arXiv preprint arXiv:1508.03326,* 2015.

[167] J. Gittins, K. Glazebrook, and R. Weber, *Multi-armed bandit allocation indices*: John Wiley & Sons, 2011.

[168] N. Galichet, "Contributions to Multi-Armed Bandits: Risk-Awareness and Sub-Sampling for Linear Contextual Bandits," Université Paris Sud-Paris XI, 2016.

[169] R. Agrawal, "Sample mean based index policies by O (log n) regret for the multi-armed bandit problem," *Advances in Applied Probability,* vol. 27, pp. 1054-1078, 1995.

[170] P. Auer, N. Cesa-Bianchi, Y. Freund, and R. E. Schapire, "The nonstochastic multiarmed bandit problem," *SIAM Journal on Computing,* vol. 32, pp. 48-77, 2002.

[171] N. Srinivas, A. Krause, S. M. Kakade, and M. Seeger, "Gaussian process optimization in the bandit setting: No regret and experimental design," *arXiv preprint arXiv:0912.3995,* 2009.

[172] E. Kaufmann, N. Korda, and R. Munos, "Thompson sampling: An asymptotically optimal finite-time analysis," in *International Conference on Algorithmic Learning Theory*, 2012, pp. 199-213.

[173] C. J. C. H. Watkins, "Learning from delayed rewards," University of Cambridge England, 1989.

[174] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*: MIT press, 1998.

[175] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-time analysis of the multiarmed bandit problem," *Machine learning,* vol. 47, pp. 235-256, 2002.

[176] J. White, *Bandit algorithms for website optimization*: " O'Reilly Media, Inc.", 2012.

[177] H. Robbins, "Some aspects of the sequential design of experiments," in *Herbert Robbins Selected Papers*, ed: Springer, 1985, pp. 169-177.

[178] M. Nowak and K. Sigmund, "A strategy of win-stay, lose-shift that outperforms tit-for-tat in the Prisoner's Dilemma game," *Nature,* vol. 364, pp. 56-58, 1993.

[179] O.-A. Maillard, R. Munos, and G. Stoltz, "A Finite-Time Analysis of Multi-armed Bandits Problems with Kullback-Leibler Divergences," in *COLT*, 2011, pp. 497-514.

[180] S. Kullback and R. A. Leibler, "On information and sufficiency," *The annals of mathematical statistics,* vol. 22, pp. 79-86, 1951.

[181] A. Garivier and O. Cappé, "The KL-UCB algorithm for bounded stochastic bandits and beyond," *arXiv preprint arXiv:1102.2490*, 2011.

[182] E. Kaufmann, O. Cappé, and A. Garivier, "On Bayesian upper confidence bounds for bandit problems," in *International Conference on Artificial Intelligence and Statistics*, 2012, pp. 592-600.

[183] Y. Freund and R. E. Schapire, "A decision-theoretic generalization of online learning and an application to boosting," in *European Conference on Computational Learning Theory*, 1995.

[184] V. Dani, S. M. Kakade, and T. P. Hayes, "The price of bandit information for online optimization," in *Advances in Neural Information Processing Systems*, 2008, pp. 345-352.

[185] S. Kale, L. Reyzin, and R. E. Schapire, "Non-stochastic bandit slate problems," in *Advances in Neural Information Processing Systems*, 2010, pp. 1054-1062.

[186] N. Cesa-Bianchi and G. Lugosi, "Combinatorial bandits," *Journal of Computer and System Sciences*, vol. 78, pp. 1404-1422, 2012.

[187] P. Auer and N. Cesa-Bianchi, "On-line learning with malicious noise and the closure algorithm," *Annals of mathematics and artificial intelligence*, vol. 23, pp. 83-99, 1998.

[188] N. Cesa-Bianchi and G. Lugosi, *Prediction, learning, and games*: Cambridge university press, 2006.

[189] P. Auer, N. Cesa-Bianchi, Y. Freund, and R. E. Schapire, "Gambling in a rigged casino: The adversarial multi-armed bandit problem," in *Foundations of Computer Science, 1995. Proceedings., 36th Annual Symposium on*, 1995, pp. 322-331.

[190] J. Komiyama, J. Honda, and H. Nakagawa, "Optimal regret analysis of thompson sampling in stochastic multi-armed bandit problem with multiple plays," *arXiv preprint arXiv:1506.00779*, 2015.

[191] Z. Wang, B. Xu, and H.-J. Zhou, "Social cycling and conditional responses in the Rock-Paper-Scissors game," *Scientific reports*, vol. 4, 2014.

[192] M. Posch, "Win Stay---Lose Shift: An Elementary Learning Rule for Normal Form Games," Santa Fe Institute1997.

[193] S. Agrawal and N. Goyal, "Analysis of Thompson Sampling for the Multi-armed Bandit Problem," in *COLT*, 2012, pp. 39.1-39.26.

[194] J. Demšar, "Statistical comparisons of classifiers over multiple data sets," *Journal of Machine learning research*, vol. 7, pp. 1-30, 2006.

[195] M. Friedman, "The use of ranks to avoid the assumption of normality implicit in the analysis of variance," *Journal of the american statistical association*, vol. 32, pp. 675-701, 1937.

[196] P. Nemenyi, "Distribution-free multiple comparisons," in *Biometrics*, 1962, pp. 263-&.

[197] E. Kaufmann, O. Cappé, and A. Garivier, "On the efficiency of Bayesian bandit algorithms from a frequentist point of view," in *Neural Information Processing Systems (NIPS)*, 2011.

[198] E. D. Sontag, "VC dimension of neural networks," *NATO ASI Series F Computer and Systems Sciences*, vol. 168, pp. 69-96, 1998.

[199] E. Keogh and A. Mueen, "Curse of dimensionality," in *Encyclopedia of Machine Learning*, ed: Springer, 2011, pp. 257-258.

[200] T. Joachims, "Text categorization with support vector machines: Learning with many relevant features," *Machine learning: ECML-98*, pp. 137-142, 1998.

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

[202] A. Krizhevsky, V. Nair, and G. Hinton, "The CIFAR-10 dataset," *online:* http://www.cs.toronto.edu/kriz/cifar.html, 2014.

[203] A. Krizhevsky, "Learning multiple layers of features from tiny images," 2009.

[204] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng, "Reading digits in natural images with unsupervised feature learning," in *NIPS workshop on deep learning and unsupervised feature learning*, 2011, p. 5.

[205] A. L. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts, "Learning word vectors for sentiment analysis," in *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*, 2011, pp. 142-150.

[206] A. M. Dai and Q. V. Le, "Semi-supervised sequence learning," in *Advances in Neural Information Processing Systems*, 2015, pp. 3079-3087.

[207] O. M. Parkhi, A. Vedaldi, and A. Zisserman, "Deep Face Recognition," in *BMVC*, 2015, p. 6.

[208] Y. Gal, "A theoretically grounded application of dropout in recurrent neural networks," *arXiv preprint arXiv:1512.05287*, 2015.

[209] R. Hadsell, S. Chopra, and Y. LeCun, "Dimensionality reduction by learning an invariant mapping," in *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, 2006, pp. 1735-1742.

[210] A. Graves and J. Schmidhuber, "Framewise phoneme classification with bidirectional LSTM and other neural network architectures," *Neural Networks*, vol. 18, pp. 602-610, 2005.

[211] J. Weston, A. Bordes, S. Chopra, A. M. Rush, B. van Merriënboer, A. Joulin, *et al.*, "Towards ai-complete question answering: A set of prerequisite toy tasks," *arXiv preprint arXiv:1502.05698*, 2015.

[212] J. Li, M.-T. Luong, and D. Jurafsky, "A hierarchical neural autoencoder for paragraphs and documents," *arXiv preprint arXiv:1506.01057*, 2015.

[213] Y. Du, W. Wang, and L. Wang, "Hierarchical recurrent neural network for skeleton based action recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1110-1118.

[214] A. Lavin, "maxDNN: an efficient convolution kernel for deep learning with maxwell gpus," *arXiv preprint arXiv:1501.06633*, 2015.

[215] P. Welinder, S. Branson, T. Mita, C. Wah, F. Schroff, S. Belongie, *et al.*, "Caltech-UCSD birds 200," 2010.

[216] S. Branson, G. Van Horn, S. Belongie, and P. Perona, "Bird species categorization using pose normalized deep convolutional nets," *arXiv preprint arXiv:1406.2952*, 2014.

[217] M.-E. Nilsback and A. Zisserman, "Automated flower classification over a large number of classes," in *Computer Vision, Graphics & Image Processing, 2008. ICVGIP'08. Sixth Indian Conference on*, 2008, pp. 722-729.

[218] D. Mishkin, N. Sergievskiy, and J. Matas, "Systematic evaluation of CNN advances on the ImageNet," *arXiv preprint arXiv:1606.02228*, 2016.

[219] D. Yu, F. Seide, G. Li, and L. Deng, "Exploiting sparseness in deep neural networks for large vocabulary speech recognition," in *Acoustics, Speech and Signal Processing (ICASSP), 2012 IEEE International Conference on*, 2012, pp. 4409-4412.

[220] S. Han, H. Mao, and W. J. Dally, "A deep neural network compression pipeline: Pruning, quantization, huffman encoding," *arXiv preprint arXiv:1510.00149*, vol. 10, 2015.

[221] F. Trovo, S. Paladino, M. Restelli, and N. Gatti, "Multi–armed bandit for pricing," in *Proceedings of the European Workshop on Reinforcement Learning (EWRL)*, 2015.

[222] N. Muralidhar. (2016). *Bandito, a Multi-Armed Bandit Tool for Content Testing*. Available: https://developer.washingtonpost.com/pb/blog/post/2016/02/08/bandito-a-multi-armed-bandit-tool-for-content-testing/

[223] T. Desautels, A. Krause, and J. W. Burdick, "Parallelizing exploration-exploitation tradeoffs in Gaussian process bandit optimization," *Journal of Machine Learning Research*, vol. 15, pp. 3873-3923, 2014.

[224] E. Hillel, Z. S. Karnin, T. Koren, R. Lempel, and O. Somekh, "Distributed exploration in multi-armed bandits," in *Advances in Neural Information Processing Systems*, 2013, pp. 854-862.

11. Appendices

APPENDIX 1 DETAILS OF THE DATA SETS AND EVALUATION 173
APPENDIX 2 VISUALIZATION ON TESTING UCI DATA SETS 199
APPENDIX 3 VISUALIZATION ON TESTING DIFFERENT DATA SETS 207
APPENDIX 4 TESTING MAB PRUNING ALGORITHMS IN REGRESSION DATA SETS 214

Appendix 1 Details of the Data sets and Evaluation

1. Evaluation data sets

1.1 Data sets for deep learning

1.1.1 ImageNet data set

ImageNet [104] is a big data set that contains millions of images based to the WordNet hierarchy, where synset (synonym set) (aka node) is the meaningful concept in WordNet that describes by words. Many images represent each node[27]. The following is the one summary of statistics of high level (main) categories[28].

Main category	Number of synset	Ave. images per synset	Total num. of images
amphibian	94	591	56K
animal	3822	732	2799K
appliance	51	1164	59K
bird	856	949	812K
covering	946	819	774K
device	2385	675	1610K
fabric	262	690	181K
fish	566	494	280K
flower	462	735	339K
food	1495	670	1001K
fruit	309	607	188K
fungus	303	453	137K
furniture	187	1043	195K
geological formation	151	838	127K
invertebrate	728	573	417K
mammal	1138	821	934K
musical instrument	157	891	140K
plant	1666	600	999K
reptile	268	707	190K
sport	166	1207	200K
structure	1239	763	946K
tool	316	551	174K
tree	993	568	564K
utensil	86	912	78K
vegetable	176	764	135K
vehicle	481	778	374K
person	2035	468	952K

[27] http://image-net.org/about-overview

[28] http://image-net.org/about-stats

1.1.2 Internet Movie Review Database (IMDB) data set

The data set serves as a benchmark for sentiment classification and contains 50,000 reviews from IMDB[205] along with their associated binary sentiment polarity labels. The data set has an even number of positive and negative reviews and split into 25k for training and 25k for testing sets. There are 50k additional unlabelled reviews for unsupervised learning. The maximum reviews for each movie is no more than 30 reviews to avoid correlated rating. Out of 10, the negative review has score less than or equal four and positive review if the score is higher than seven. Then reviews with neural rating is not included in both training and testing sets. In contrast of that unlabelled data is included positive, negative, and neural reviews[29].

1.1.3 Reuters newswire topic classification task

This data set made by Reuters Ltd for research. The data set contains large of news stories. In this thesis, we use 8982 for training (9083 train, 899 validate) and 2246 for testing[30].

1.1.4 Mixed National Institute of Standards and Technology (MNIST) data set

The MNIST data set is a collection of handwritten digits which contains 60,000 examples of training set and 10,000 examples for test set. The digits have been size-normalized and cantered in 28 by 28 image [31]. The following diagram shows the sample of this data set where it shows the numbers 5, 0, 4 and 1. However, the first digit is confused between 3 and 5.

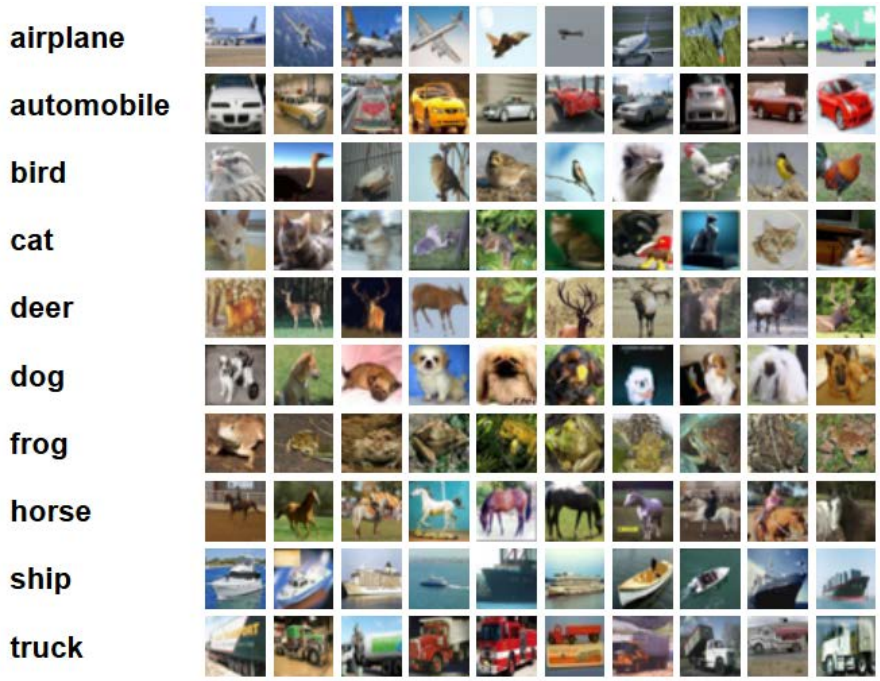
[29] http://ai.stanford.edu/~amaas/data/sentiment/

[30] http://trec.nist.gov/data/reuters/reuters.html

[31] http://yann.lecun.com/exdb/mnist/

1.1.5 The Cifar-10 data set

CIFAR-10 data set is a collection of 32x32 colour images in 10 different classes. The data set splits into two sets. The first set is 50,000 images for training and the other is 10,000 for testing [32]. The following diagram shows the classes in the data set, as well as 10 random images from each class[32].



1.1.6 The Cifar-100 data set

Cifar-100 is the same as cifar-10 data set with different of the number of classes where here there is 100 classes. Each class has 600 images where splits to two groups, the first group is 500 images for training and the rest for testing. As shown in the following table, the data set is divided to 20 super classes and each class has five class.[32].

[32] https://www.cs.toronto.edu/~kriz/cifar.html

Superclass	Classes
aquatic mammals	beaver, dolphin, otter, seal, whale
fish	aquarium fish, flatfish, ray, shark, trout
flowers	orchids, poppies, roses, sunflowers, tulips
food containers	bottles, bowls, cans, cups, plates
fruit and vegetables	apples, mushrooms, oranges, pears, sweet peppers
household electrical devices	clock, computer keyboard, lamp, telephone, television
household furniture	bed, chair, couch, table, wardrobe
insects	bee, beetle, butterfly, caterpillar, cockroach
large carnivores	bear, leopard, lion, tiger, wolf
large man-made outdoor things	bridge, castle, house, road, skyscraper
large natural outdoor scenes	cloud, forest, mountain, plain, sea
large omnivores and herbivores	camel, cattle, chimpanzee, elephant, kangaroo
medium-sized mammals	fox, porcupine, possum, raccoon, skunk
non-insect invertebrates	crab, lobster, snail, spider, worm
people	baby, boy, girl, man, woman
reptiles	crocodile, dinosaur, lizard, snake, turtle
small mammals	hamster, mouse, rabbit, shrew, squirrel
trees	maple, oak, palm, pine, willow
vehicles 1	bicycle, bus, motorcycle, pickup truck, train
vehicles 2	lawn-mower, rocket, streetcar, tank, tractor

1.1.7 The Street View House Numbers (SVHN) Data set

The SVHN data set [204] is a collection of 32x32 read world over 600,000 digit images. The data set is acquired from house numbers in Google Street View images. The following diagram shows the sample of cropped digits[33].

[33] http://ufldl.stanford.edu/housenumbers/

1.1.8 The bAbI Data set

The bAbI data set [101, 211] contains 20 different tasks where each task provides the training and testing data set. Task 6 has been used in this thesis[34]. The following diagram shows sample of this task as cited by Weston et al.

Task 6: Yes/No Questions
John moved to the playground.
Daniel went to the bathroom.
John went back to the hallway.
Is John in the playground? A:no
Is Daniel in the bathroom? A:yes

1.1.8 102 flower Data set

The data set [217] contain 102 classes of common flower in the UK. Each class contain from 40 to 258 images. The following is sample of the data set.

Category	#ims	Category	#ims	Category	#ims
 alpine sea holly	43	 buttercup	71	 fire lily	40
 anthurium	105	 californian poppy	102	 foxglove	162
 artichoke	78	 camellia	91	 frangipani	166
 azalea	96	 canna lily	82	 fritillary	91
 ball moss	46	 canterbury bells	40	 garden phlox	45
 balloon flower	49	 cape flower	108	 gaura	67
 barbeton daisy	127	 carnation	52	 gazania	78

[34] https://research.facebook.com/research/babi/

1.1.9 Birds 200 Data set

The data set [215] contain 200 bird categories and 6,033 images mostly from North America and the following is sample of the data set.

1.2 Data sets for shallow neural networks in classification task

1.2.1 banknote authentication Data Set

The data set contains 1372 examples of four continues feature (variance of wavelet, skewness of wavelet, kurtosis of wavelet and entropy of images) and binary class. However, the data set is extracted from images that were taken from genuine and forged banknote-like specimens [35].

1.2.2 Blood Transfusion Service Centre Data Set

"The database of Blood Transfusion Service Centre in Hsin-Chu City in Taiwan." [36]. The data set contains 748 examples, four features (R, F, M and T) and binary classes.

1.2.3 Credit Approval Data Set

This data concerns 690 examples of credit card applications [37] and contains 15 features and binary classes.

---

[35] https://archive.ics.uci.edu/ml/data sets/banknote+authentication

[36] https://archive.ics.uci.edu/ml/data sets/Blood+Transfusion+Service+Center

[37] https://archive.ics.uci.edu/ml/data sets/Credit+Approval

1.2.4 Haberman's Survival Data Set

The data set contains 306 examples of cases from a study that was conducted between 1958 and 1970 at the University of Chicago's Billings Hospital on the survival of patients who had undergone surgery for breast cancer[38]. The data set has three features and binary class.

1.2.5 Liver Disorders Data Set

The data set contains 345 examples from BUPA Medical Research Ltd. Database[39]. The data set has six features and binary class.

1.2.6 MAGIC Gamma Telescope Data Set

The data set contains 19020 examples of MC generated to simulate registration of high energy gamma particles in a ground-based atmospheric Cherenkov gamma telescope using the imaging technique[40]. The data set has 10 features and binary class.

1.2.7 Mammographic Mass Data Set

This data set contains 961 examples of a mammographic mass lesion from BI-RADS attributes and the patient's age. The data set has five features and binary class [41].

1.2.8 MONK's Problems Data Set

The data set contains 432 examples of MONK's problem. The data set has six features and binary class**[42].**

1.2.9 Connectionist Bench (Sonar, Mines vs. Rocks) Data Set

The data contains 208 examples to discriminate between sonar signals bounced off a metal cylinder and those bounced off a roughly cylindrical rock. The data set has 59 features and binary class[43].

---

[38] https://archive.ics.uci.edu/ml/data sets/Haberman's+Survival

[39] https://archive.ics.uci.edu/ml/data sets/Liver+Disorders

[40] https://archive.ics.uci.edu/ml/data sets/MAGIC+Gamma+Telescope

[41] https://archive.ics.uci.edu/ml/data sets/Mammographic+Mass

[42] https://archive.ics.uci.edu/ml/data sets/MONK's+Problems

[43] https://archive.ics.uci.edu/ml/data sets/Connectionist+Bench+%28Sonar,+Mines+vs.+Rocks%29

1.2.10 Spambase Data Set

This data set contains 4601 examples to detect an email either it is spam or not. The data set has 56 features and binary class[44].

1.2.11 SPECTF Heart Data Set

The data set contains 267 examples to describes diagnosing of cardiac Single Proton Emission Computed Tomography (SPECT) images. The data set contains 44 features and binary class[45].

1.2.12 Tic-Tac-Toe Endgame Data Set

This database contains 958 examples to encodes the complete set of possible board configurations at the end of tic-tac-toe games. The data set has nine feature and binary class[46].

1.2.13 Pima Indians Diabetes Data Set

This data set contains 768 examples of females' patients at least 21 years old of Pima Indian heritage. The data set has nine features and binary class[47].

1.2.14 Breast Cancer Wisconsin Data Set

This data set contains 569 examples to diagnostic Wisconsin breast cancer. The data set has 32 features and binary class[48].

1.2.15 Adult Data Set

This data set contains 48842 examples to predict whether income exceeds $50K/yr. based on census data. The data set has 14 features and binary class[49].

---

[44] https://archive.ics.uci.edu/ml/data sets/Spambase

[45] https://archive.ics.uci.edu/ml/data sets/SPECTF+Heart

[46] https://archive.ics.uci.edu/ml/data sets/Tic-Tac-Toe+Endgame

[47] https://archive.ics.uci.edu/ml/data sets/Pima+Indians+Diabetes

[48] https://archive.ics.uci.edu/ml/data sets/Breast+Cancer+Wisconsin+%28Diagnostic%29

[49] https://archive.ics.uci.edu/ml/data sets/Adult

1.2.16 Hill-Valley Data Set

This data set contains 606 examples of Hill Valley data set. The data set has 100 features and binary class[50].

1.2.17 Titanic data set

This data set contains 1309 examples with 11 features and binary class. The data set represents the sinking of the RMS[51].

1.2.18 Wine data set

This data set contains 178 examples of a chemical analysis of wines grown in the same region in Italy. It has 12 features and three classes[52].

1.2.19 Heart Disease Data Set

The data set contains 303 examples of the presence of heart disease in the patient. It has 13 features and binary class[53].

1.2.20 Iris Data Set

This popular data set contains 150 examples of types of iris plant. It has three features and three classes[54].

1.2.21 Car Evaluation Data Set

This data set has 1728 examples with six features and four classes[55].

---

[50] https://archive.ics.uci.edu/ml/data sets/Hill-Valley

[51] https://www.kaggle.com/c/titanic

[52] https://archive.ics.uci.edu/ml/data sets/Wine

[53] https://archive.ics.uci.edu/ml/data sets/Heart+Disease

[54] http://archive.ics.uci.edu/ml/data sets/Iris

[55] https://archive.ics.uci.edu/ml/data sets/Car+Evaluation

1.2.22 Abalone Data Set

This data set contains 4177 examples for predicting the age of abalone from physical measurements. It has eight features and three classes[56].

1.2.23 Poker Hand Data Set

This data set contains 1015010 examples of a hand consisting of five playing cards drawn from a standard deck of 52. It has 10 features and 10 classes of poker hand[57].

1.2.24 Glass Identification Data Set

This data set contains 214 examples of glass types. It has 10 features with seven classes "class types"[58].

1.2.25 Wine Quality Data Set

This data set contains 4898 examples of the quality of wine. It has 11 features with 10 types of classes[59].

1.2.26 Face data set

The data set contains over 13,000 images of faces (examples) collected from the web where each face has been labelled with the name of the person pictured[60]. In this thesis, we use the top 5 most represented people in the data set which make the samples contains 1288 examples, 1850 features and seven classes.

---

[56] https://archive.ics.uci.edu/ml/data sets/Abalone

[57] https://archive.ics.uci.edu/ml/data sets/Poker+Hand

[58] https://archive.ics.uci.edu/ml/data sets/Glass+Identification

[59] https://archive.ics.uci.edu/ml/data sets/Wine+Quality

[60] http://vis-www.cs.umass.edu/lfw/

1.3 Data sets for shallow neural networks in regression task

1.3.1 Housing data set

The data set contains 506 examples of concerns housing values in suburbs of Boston. It has 13 features[61].

1.3.2 Airfoil Self-Noise Data Set

The data set contains 1503 examples of a series of aerodynamic and acoustic tests from NASA. It has five features[62].

1.3.3 Auto MPG Data Set

This data set contains 398 examples and it has eights features[63].

1.3.4 Computer Hardware Data Set

The data set contains 209 examples of computer's hardware and nine features[64].

1.3.5 Concrete Compressive Strength Data Set

The data set contains 1030 of concrete and it has eight features[65].

1.3.6 Parkinsons Telemonitoring Data Set

This data set contains 5875 examples of Parkinson's disease. It has 21 features.[66].

[61] https://archive.ics.uci.edu/ml/data sets/Housing

[62] https://archive.ics.uci.edu/ml/data sets/Airfoil+Self-Noise

[63] https://archive.ics.uci.edu/ml/data sets/Auto+MPG

[64] https://archive.ics.uci.edu/ml/data sets/Computer+Hardware

[65] https://archive.ics.uci.edu/ml/data sets/Concrete+Compressive+Strength

[66] https://archive.ics.uci.edu/ml/data sets/Parkinsons+Telemonitoring

2. List of classifiers (estimators) used to compare with MAB pruning algorithms

Two kinds of models have been built to compare with MAB pruning algorithms either models for classification or regression. First, the following list is the well-known algorithms used to build different classifiers so we compare their results with our pruned methods.

- K nearest neighbour (KNN) classifier.
- Support vector machines (SVM) classifier, here we used linear SVM and rbf kernel.
- Decision tree (DT) classifier, we used two different decision trees which is CART (classification and regression trees) with gini impurity and C5.0 with entropy impurity.
- Bagging method, we used bagging with both decision trees and KNNs.
- Random forest (RF), again we used with both decision trees and KNNs.
- Adaptive Boosting (AdaBoost) with only decision trees.
- Naïve Bayes
- Linear discriminant analysis (LDA) classifier.
- Quadratic discriminant analysis (QDA) classifier.
- Gaussian process classifier
- LightGBM
- Logistic regression
- Xgboost
- Neural network (NN), which the original model we pruned it. We used one of nonlinearity activations function according the results in cross validation.

While the following list is the well-known regression's algorithms that compare our results against them.

- Ordinary least square
- Linear regression
- Lasso regression
- Bayesian Ridge regression
- Kernel ridge regression
- Decision tree (DT) regressor, decision tree with mean squared error.
- Gradient Boosting (Xgboost), with decision tress.

- Neural network (NN), which the original model we pruned it. We used mean square error activation loss function.
- Bagging method, we used bagging with decision trees.
- Adaptive Boosting (AdaBoost) with only decision trees.
- Support vector machines (SVM) regression.

1. Building the models

Mostly, there is no pretrained models available for small data sets so we need to build a neural network model then we can prune it later. We want to make sure that the model we have built is one of the best model on the data set. So, many steps of recipe we need to follow to guarantee we prune one of the best model and our pruned models can work better or at least there is no big loss in the performance where retraining the pruned models are needed. We applied this scenario to finding best neural network models and we did the same on the rest of machine learning algorithms. So, many steps of building best models have been tested to guarantee that we have the best model of each machine learning model. The following steps of recipe we follow to guarantee that we have the best for all models.

1.1 Visualize the data

The first step is visualizing the data and understand it. The goal of visualization is that checking if there is missing data, to know the important features to each model, to find if there is correlation between features then later can cope of the features' correlations and many others. Pandas package[67] (Python Data Analysis Library) has been used to do the visualization with some other Python's packages (matplotlib[68], ggplot[69], seaborn[70] and Bokeh[71]). Figure 11.1 shows an example of the plotting data set to see the relation between the features and the rest is shown in thesis online code [72].

[67] http://pandas.pydata.org/

[68] http://matplotlib.org/

[69] http://ggplot.yhathq.com/

[70] https://stanford.edu/~mwaskom/software/seaborn/

[71] http://bokeh.pydata.org/en/latest/

[72] https://github.com/SalemAmeen/chapter-five

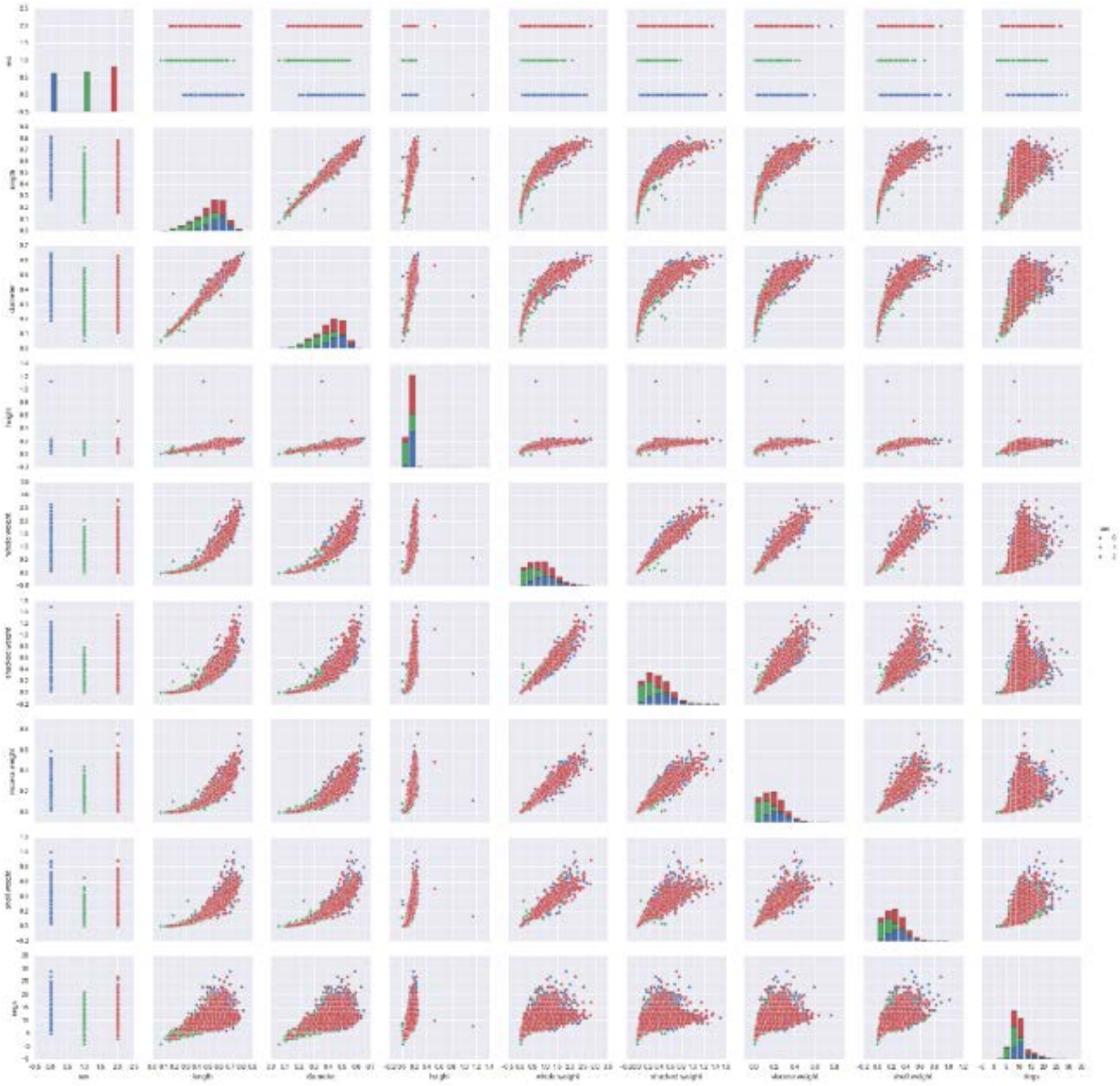


Figure 11.1: Visualization of abalone data.

We use many techniques to find the important features like decision tree, p-value, and correlation to the target. Figure 11.2 shows some plots of important features where decision tree used in some data sets and the rest is available on the thesis online code [72].

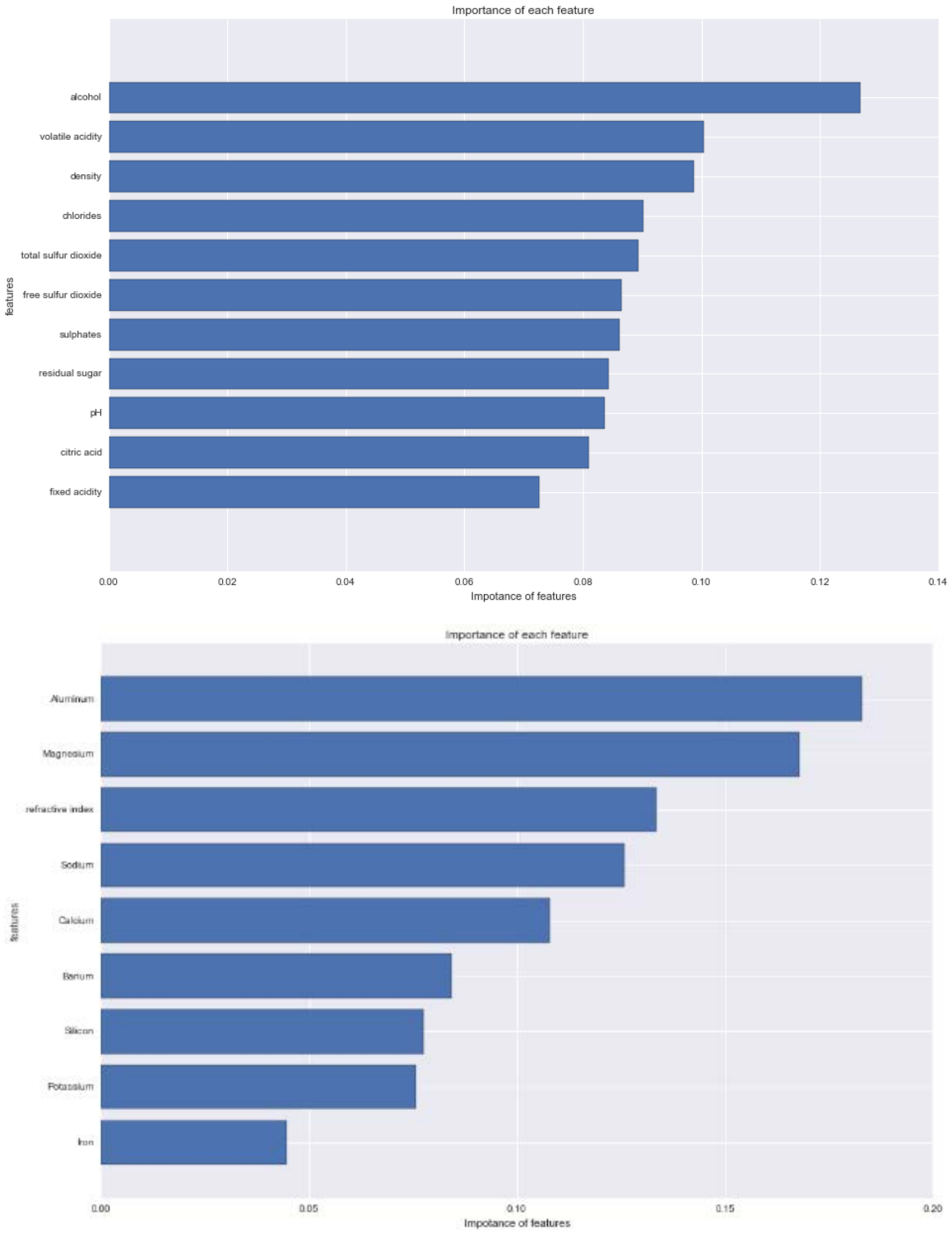


Figure 11.2: Important Features: on the top Wine quality data set and on the bottom glass data set.

The main goal to see if the feature important or not. In some cases, we add many features and use PCA to get smaller dimensions.

Finally, to see the correlation (Pearson, Kendall, and Spearman) between features or the performance of adding new features to the model three kind of correlation have been checked all data set (except that for deep learning). Figure 11.3 shows some plots of those correlation and the rest available on the thesis online code [72].

Pearson correlation is firstly used to find the linear correlation between variables to avoid collinearity between variables then if the data is qualitative other correlation is used to extract the correlation between variables.

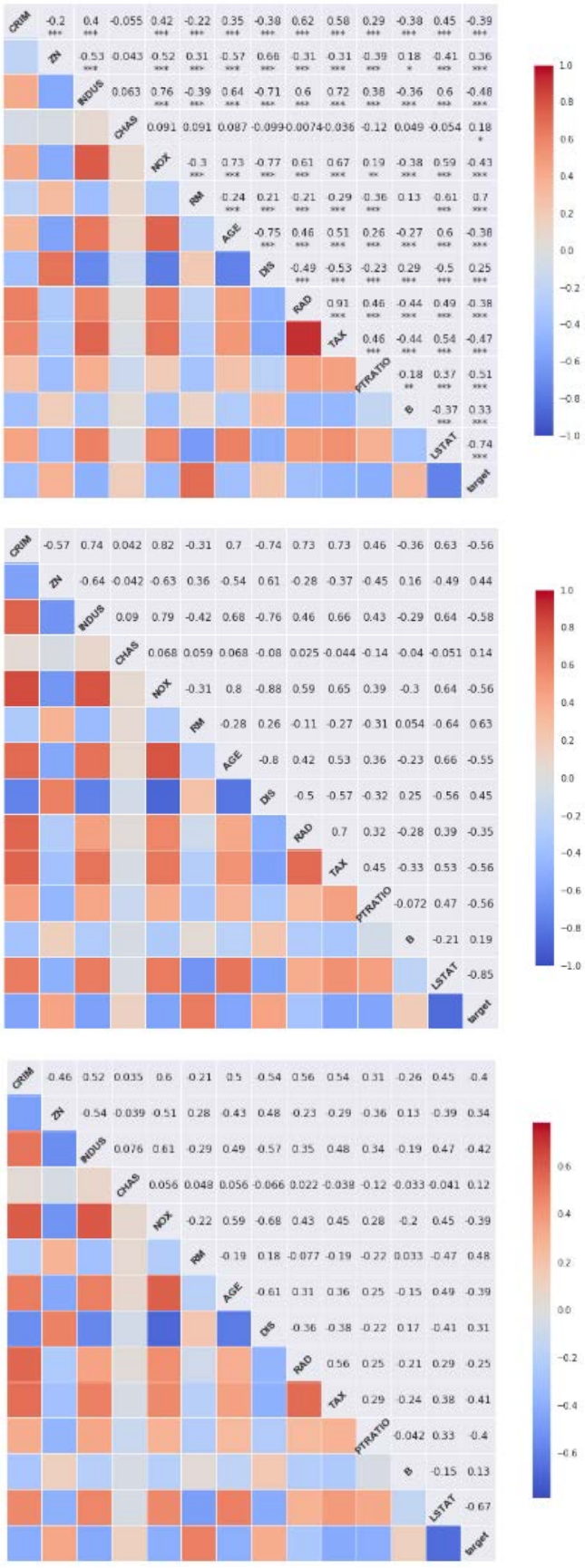


Figure 11.3: The correlation between variables from left to right, Pearson correlation, Spearman correlation Kendall correlation.

1.2 Operations on Data set

There are many operations that apply to the data set before start building the models. The first step is dividing the data to groups. The first group is the training data which contains 60% of the data for building the models and 20% for validation. The 20% rest is for testing, this data set kept unseen until we finish prune the models and compare the results with the original neural network and with other machine learning models. While if the data is small, we use cross validation to build the models. However, in data set like ImageNet, MNIST and some others, we just divide it into two groups, one for training and the other for the validate the results. Whilst in most small data sets, we use 10 cross validations to build the models. Appendix 1 shows the all data sets that used in this thesis. Scikit-learn (sklearn) machine learning library [73] is used to split the data, clean the data, adding new features to the data and other pre-processing to the small data. All those operations are done to the training data set then the needed transformations like computing the mean and standard deviation when normalize the training data are saved to use them on validation and testing time. Transforming non-numerical features to numerical or filling missing data either by computing mean, median or predict it using regression or decision trees is done on training data set. However, we guarantee that all the models use the same training and testing data set to avoid any bias to specific algorithm. However, most the data sets that made for deep learning are already split into training and validation/testing.

1.3 Feature selection

Two methods used to select the features to build all models. The first method is selecting the best important features that effect the output of the model. The second method, we use principle competent analysis (PCA) to reduce the dimension when the data has high dimension features such as face data set. Figure 11.4 shows the projected data from PCA on face data set. However, sometimes we used both methods specially when new features are generated or transforming the object features to numerical features. The transformation introduces many new features and sometimes lead to cruse of dimensionality. First, we apply PCA to all the features then choose the best some principles. Second, we choose the best features from the original features of the data. Finally, we concatenate them together to get the new features. Nevertheless, cross validation is used to choose which method is preferred

[73] http://scikit-learn.org/stable/#

and the number of features that are suitable for the model. Finally, those extracted features applied to all classifier.

Figure 11.4: PCA on face data set. The 12 photos on the left show samples from the data while the 12 on the right show images after applying PCA.

1.4 Validation curve

Using Sklearn library we plot the influence of a single hyperparameter on the training score and the validation score for two reasons. The first reason, some hyperparameter have continuous and unlimited (like k in KNN) space so finding the best combination with other hyperparameters is extremely expensive. Then, we use this method to bound the single hyperparameter then later we can use this bounded range with other hyperparameters to find the best combination of hyperparameters to the model. If the hyperparameter has small options (like the type of algorithm in KNN (‘auto,’ ‘ball_tree,’ ‘kd_tree,’ ‘brute’), we did not use this method. The second reason, to find out whether the model is suffering from overfitting or underfitting for some hyperparameter values. Figure 11.5 shows some plot of some parameters in a model and the rest can be find in the thesis online code [72].

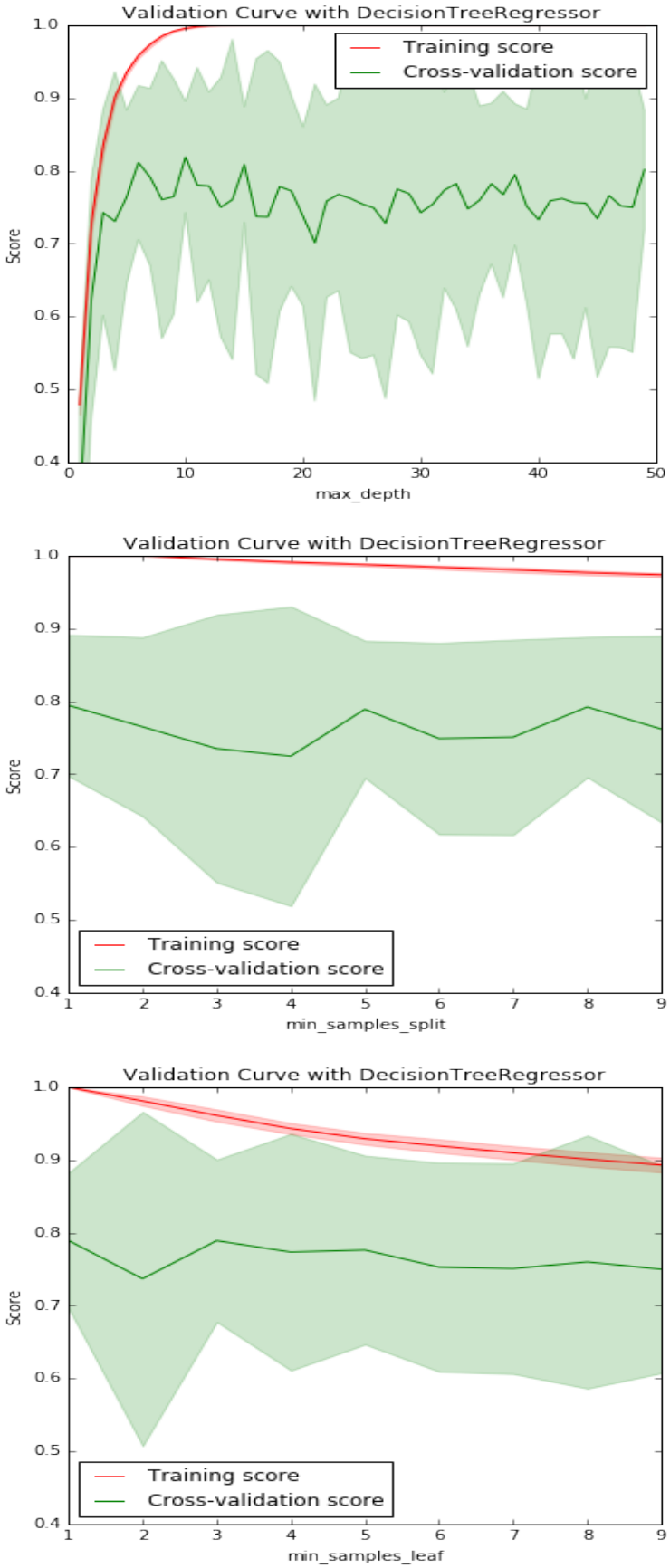


Figure 11.5: Hyperparameters of the decision tree on Boston house data set.

From Figure 11.5, the diagram on the top shows that the max depth of the tree is preferable greater than 10 while the diagram on the middle shows the minimum samples split is less than 3 and finally the diagram on the bottom shows that minimum sample leaf is less than 2.

2. Searching in parameters space

After bounded the continues parameters and the parameters that have many values, two different searching methods used to find the best combination of hyperparameters. First one, we use the random searching where we set a range (according to validation curve method) for each hyperparameter and let the algorithm find the best combination. The second method, we set the combinations of some expected good values (where we expected the optimal combination) of parameters then grid search used to find the best combination.

Finally, we print the list of combinations of hyperparameters where we think the optimal combination will be one of them. Generally, the top on the list will be best combination but we tried many of them using cross validation. Figure 11.6 shows the example of using searching of the parameters and the rest can be found on the thesis online code [72].

```
Hyperparameter optimization using GridSearchCV...
Parameters with rank: 1
Mean validation score: 0.8227 (std: 0.1063)
Parameters: {'n_estimators': 8, 'loss': 'linear'}
 Parameters with rank: 2
Mean validation score: 0.8202 (std: 0.1180)
Parameters: {'n_estimators': 8, 'loss': 'square'}
 Parameters with rank: 3
Mean validation score: 0.8178 (std: 0.1108)
Parameters: {'n_estimators': 9, 'loss': 'linear'}
 Parameters with rank: 4
Mean validation score: 0.8103 (std: 0.0909)
Parameters: {'n_estimators': 8, 'loss': 'exponential'}
 Parameters with rank: 5
Mean validation score: 0.8052 (std: 0.0935)
Parameters: {'n_estimators': 7, 'loss': 'square'}
 ++++++++++++++++++++++++++++++++++++++++++++++++++
 Hyperparameter optimization using RandomizedSearchCV
Parameters with rank: 1
Mean validation score: 0.8468 (std: 0.0957)
Parameters: {'n_estimators': 100, 'loss': 'square'}
 Parameters with rank: 2
Mean validation score: 0.8435 (std: 0.0935)
Parameters: {'n_estimators': 20, 'loss': 'square'}
 Parameters with rank: 3
Mean validation score: 0.8162 (std: 0.1297)
Parameters: {'n_estimators': 10, 'loss': 'linear'}
 Parameters with rank: 4
Mean validation score: 0.7018 (std: 0.0992)
Parameters: {'n_estimators': 1, 'loss': 'linear'}
 Parameters with rank: 5
Mean validation score: 0.6801 (std: 0.1947)
Parameters: {'n_estimators': 1, 'loss': 'square'}
```

Figure 11.6: Finding best combination of parameters for Adaboost.

From the Figure 11.6, the parameters generated using random search has better results. However, we are trying all the configurations on the data set.

3. Learning curve

The main goal of learning curve is to balance the bias and variation trade-off of the models. In other words, it is a tool that is used to find out how much we benefit from adding more training data and whether the model suffers more from a variance error or a bias error. However, in those experiments we cannot add more data to the models as the data is limited by the resources and the goal is to build the best model on those data sets. Therefore, we used learning curve to find the best ranked combinations of hyperparameters that are found using searching on parameters space in the previous section. Figure 11.7 shows validation curve of a model while the rest can be found on the thesis online code [72].

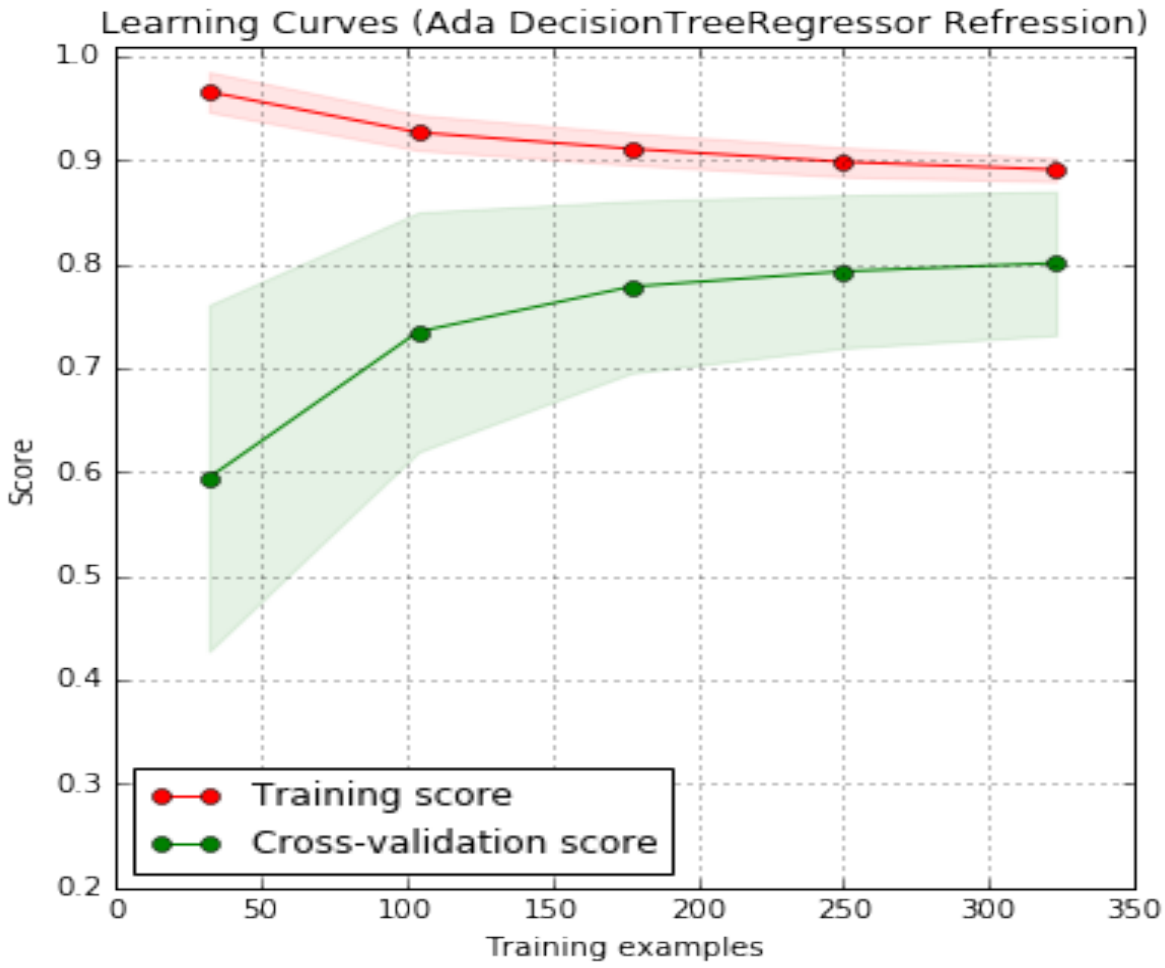


Figure 11.7: Adaboost on Boston house data set.

In the Figure 11.7, the score of the validation data increase as the training data set increase in contrast of that the training score is decrease when the training data is increase. In general, the model can learn the function.

4. Training the models

After we make sure there is no bias and various on the models by using validation curve, we start training the models one by one on training data and monitoring the performance using cross validation. However, if the validation performance is not good we retrain them on other combination until we find the best validation performance. We used various metrics like accuracy, f1 score, recall, precision, support, confusion matrix, ROC and others are used to see the performance of validation data. After trained neural network model and other machine learning models, we save the trained models to test them with our pruning models and see the performance of our pruned models comparing to original neural networks model before prune and the other machine learning models. On the thesis, online code [72], there are many metrics on training and validation data sets.

With the binary data sets, we compute and plot the receiver operating characterize (ROC) in each model. Figure 11.8 shows sample of the plotting of ROC in adult data set and the rest is available on the thesis online code [72].

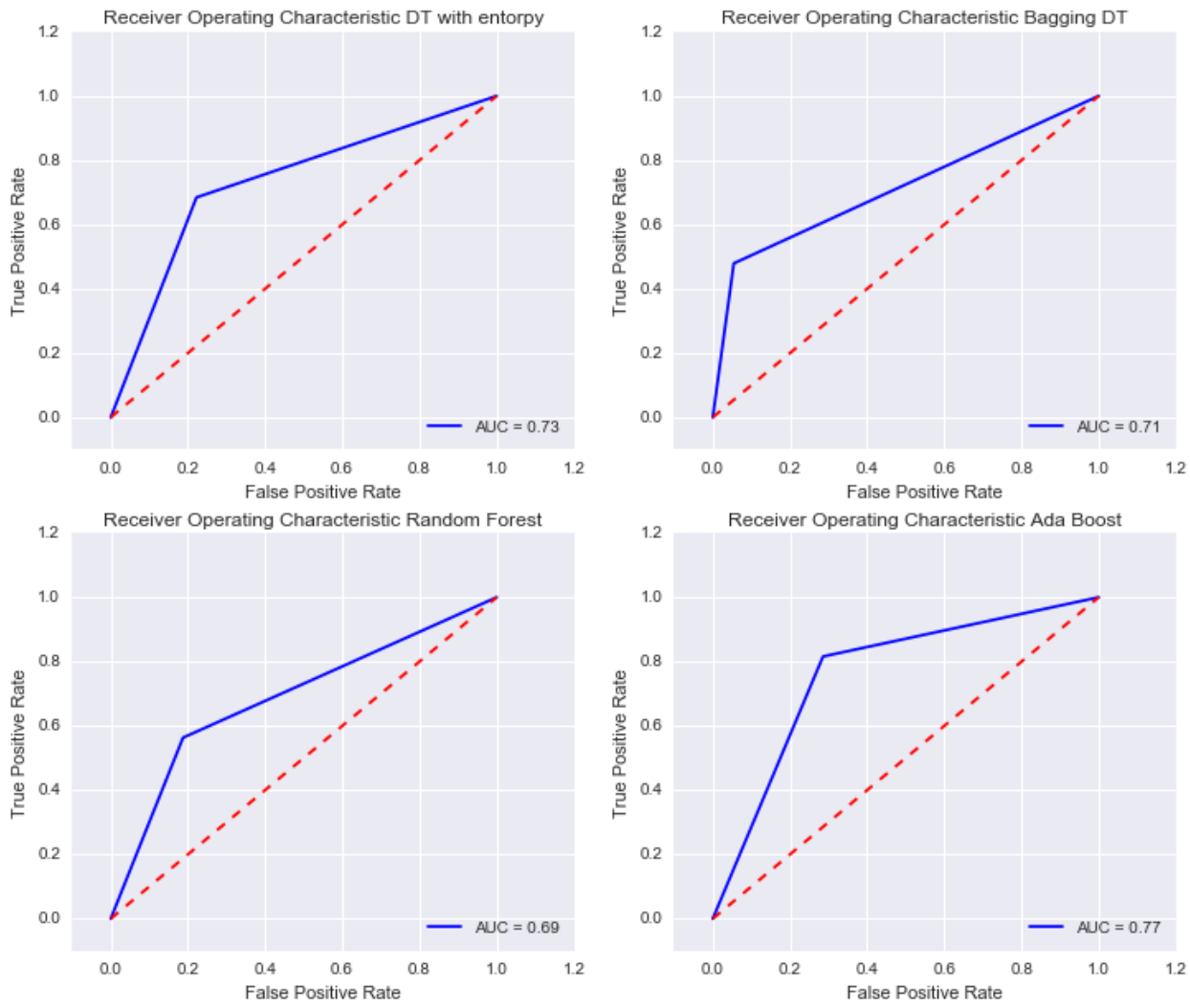
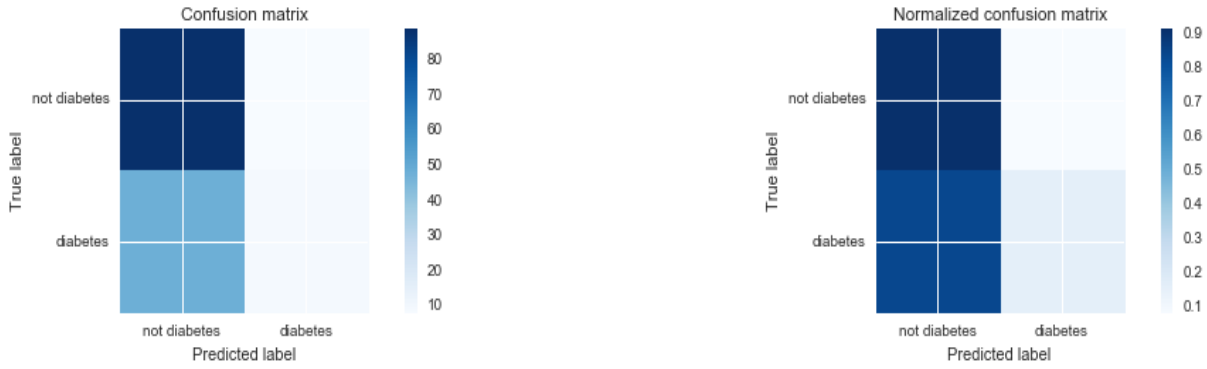
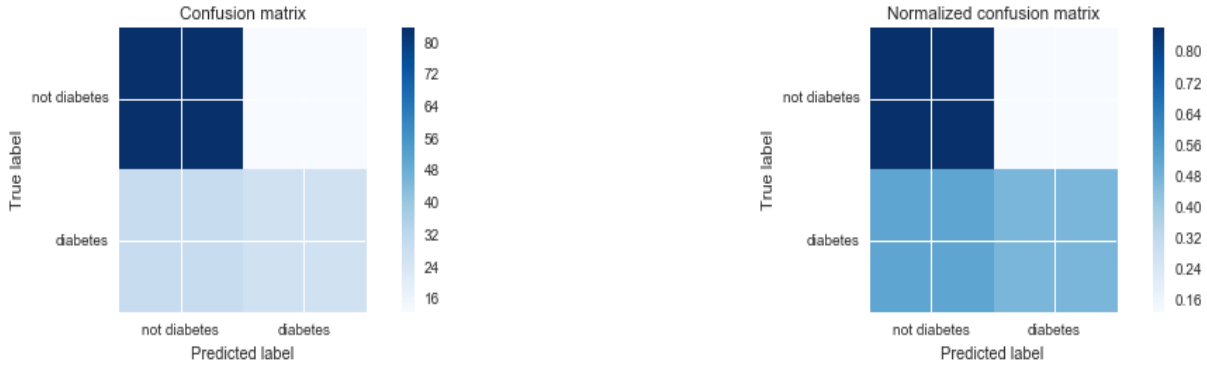


Figure 11.8: ROC on adult data set.

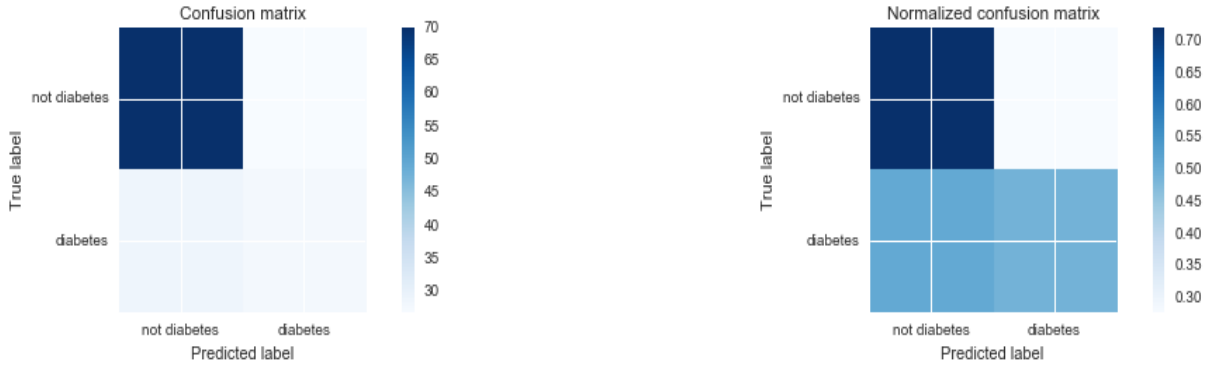
Finally, we plot the confusion matrix on all testing data sets. Figure 11.9 shows sample the confusion matrix on pima data set. We can see an improvement of the performance of the pruned models over the original neural networks models.



Linear SVM



SVM

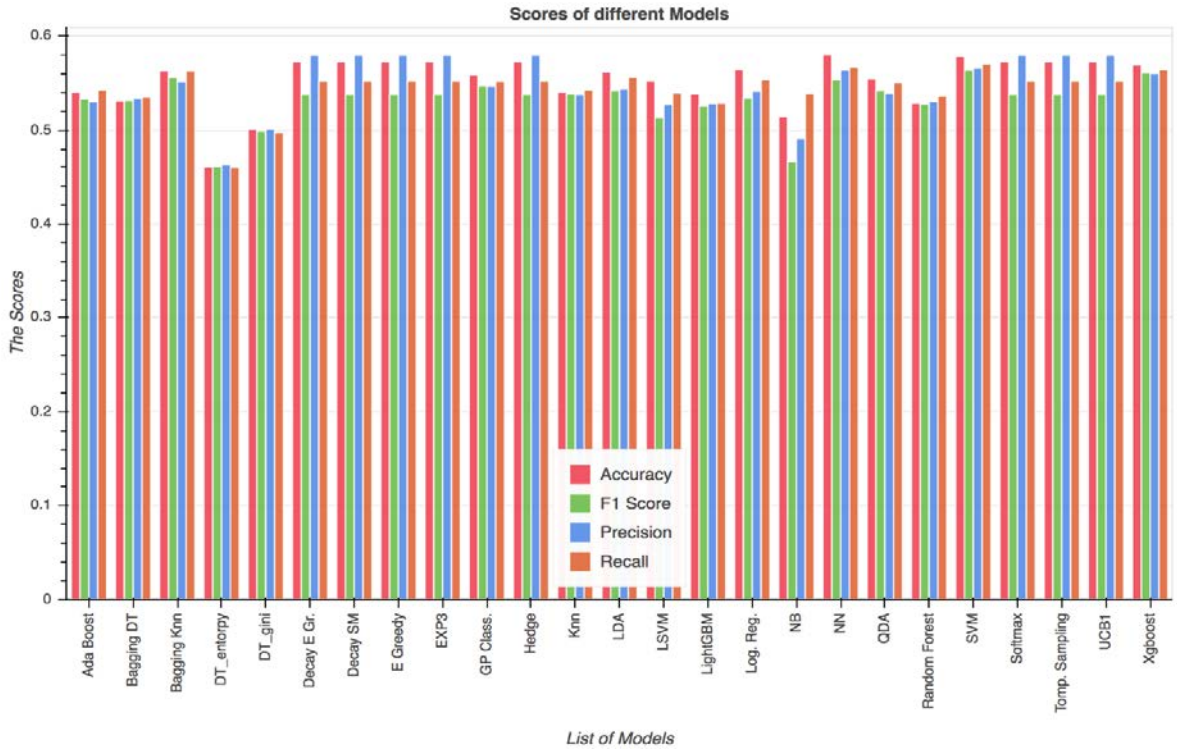


Decision Tree (CART)

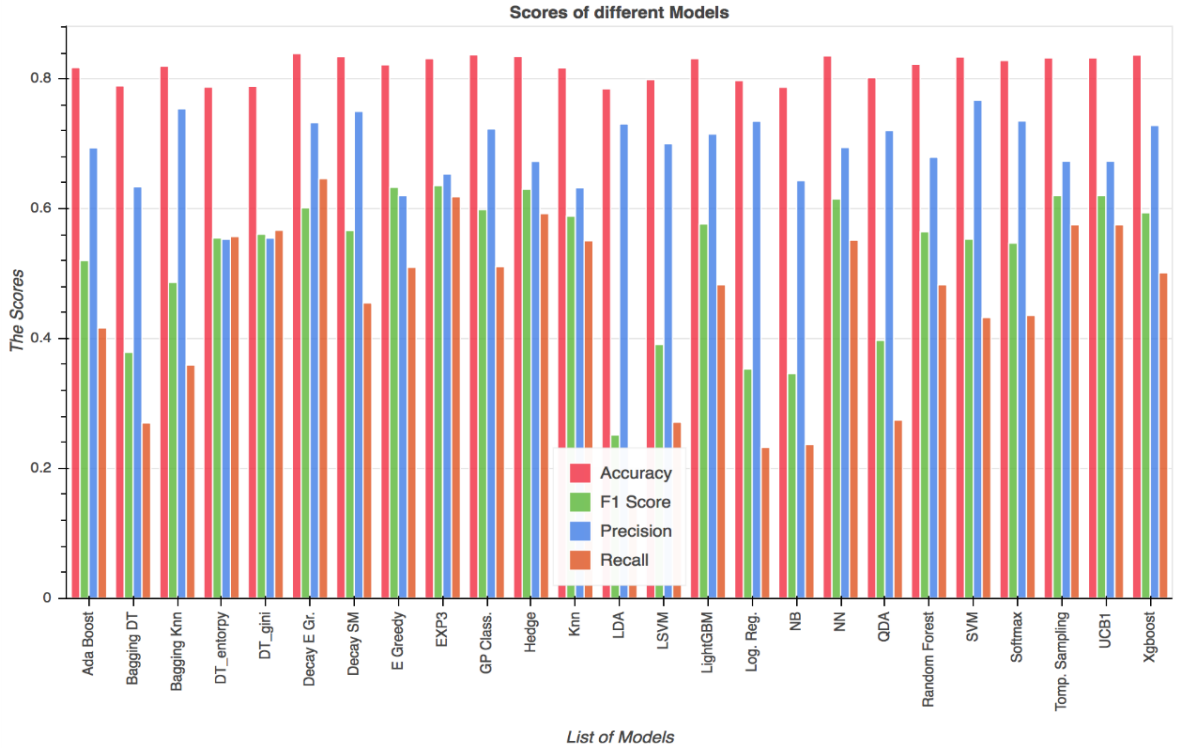
Decision Tree (C5.0)

Figure 11.9: Some confusion matrices for the pima data set

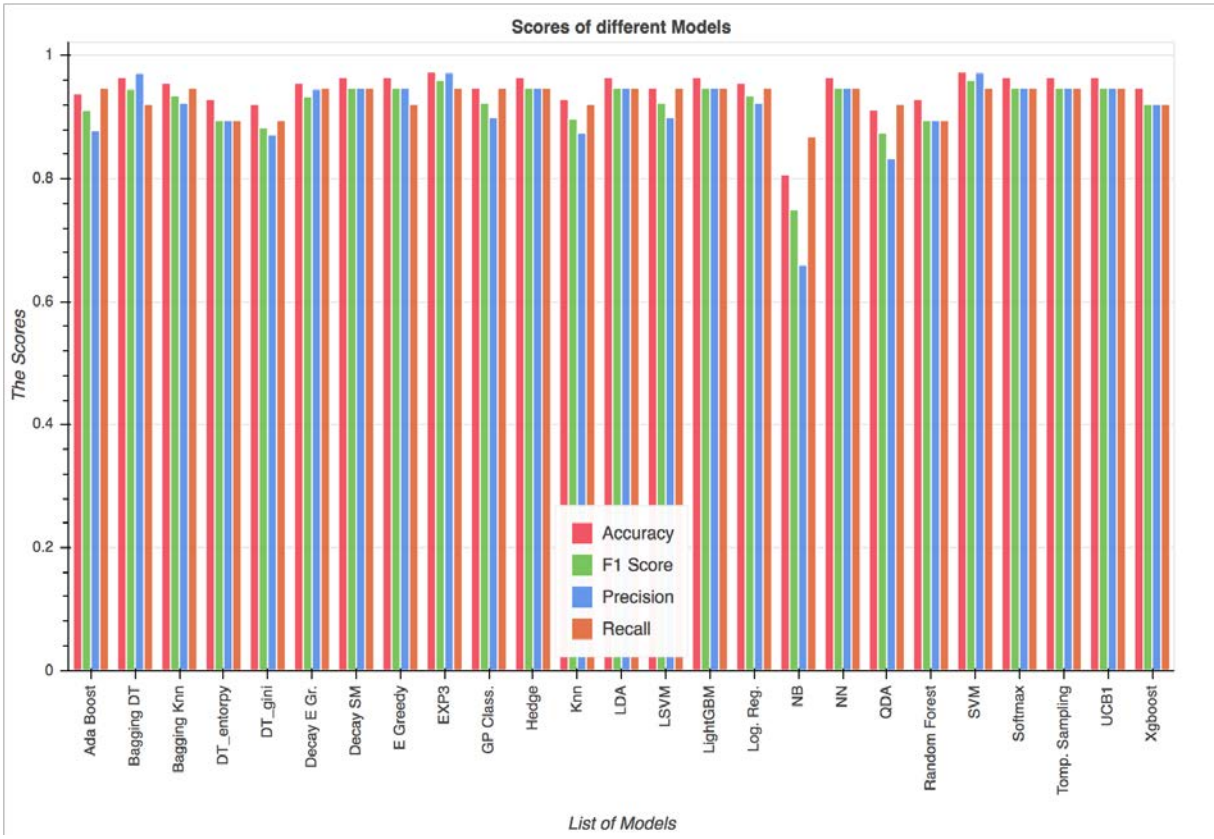
Appendix 2 Visualization on testing UCI Data sets



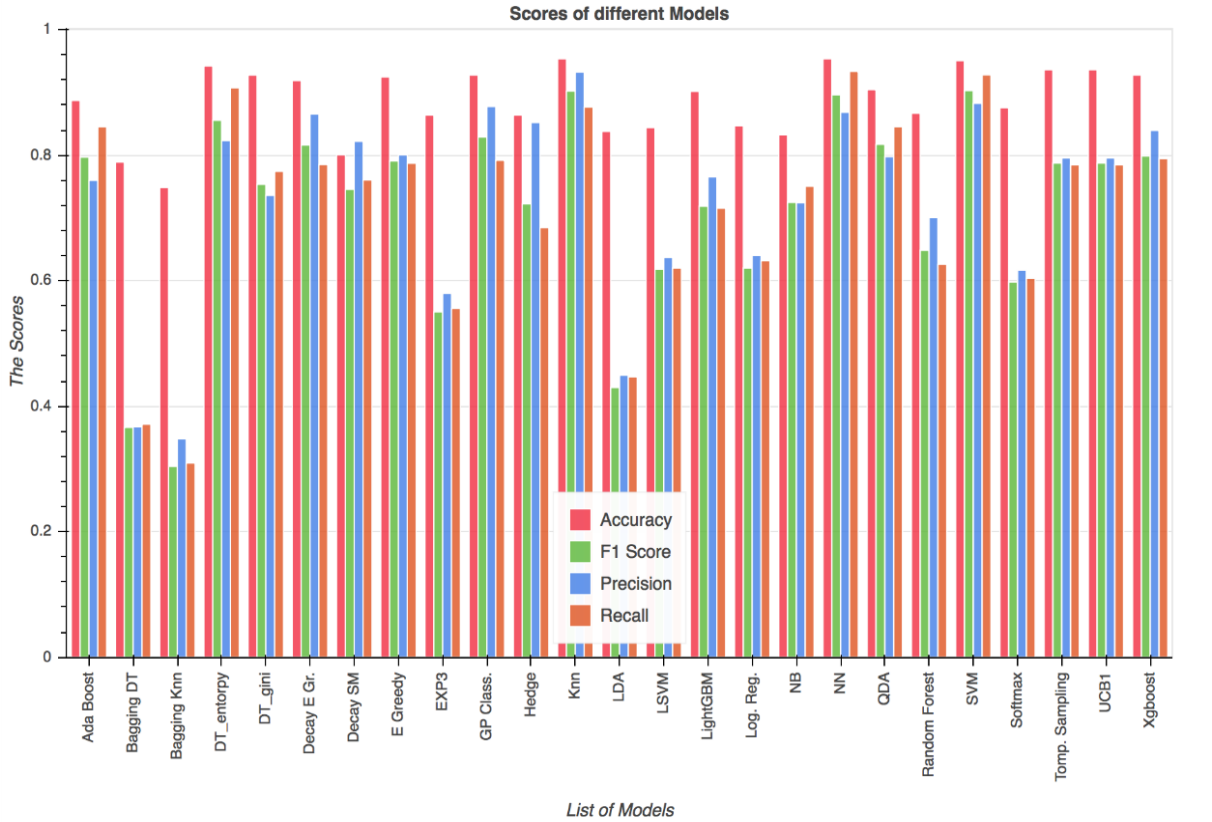
Abalone data set



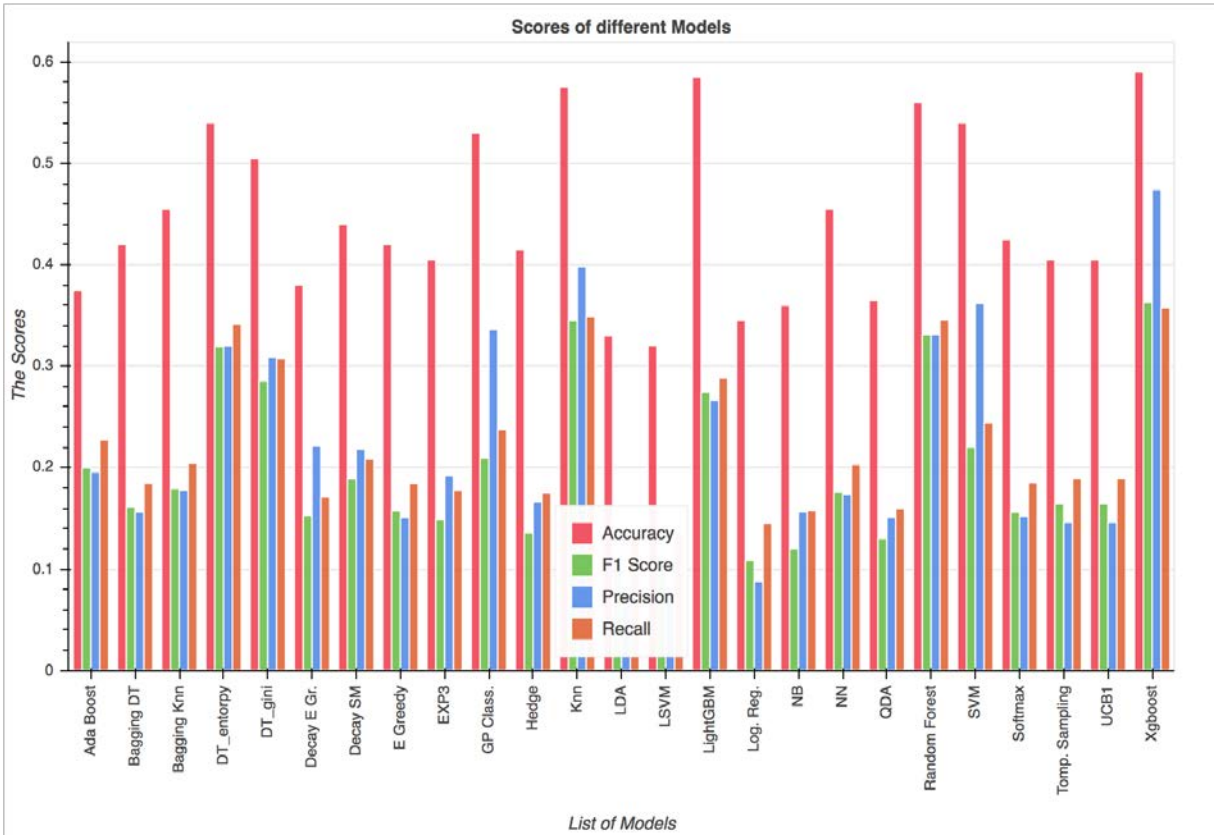
Adult data set



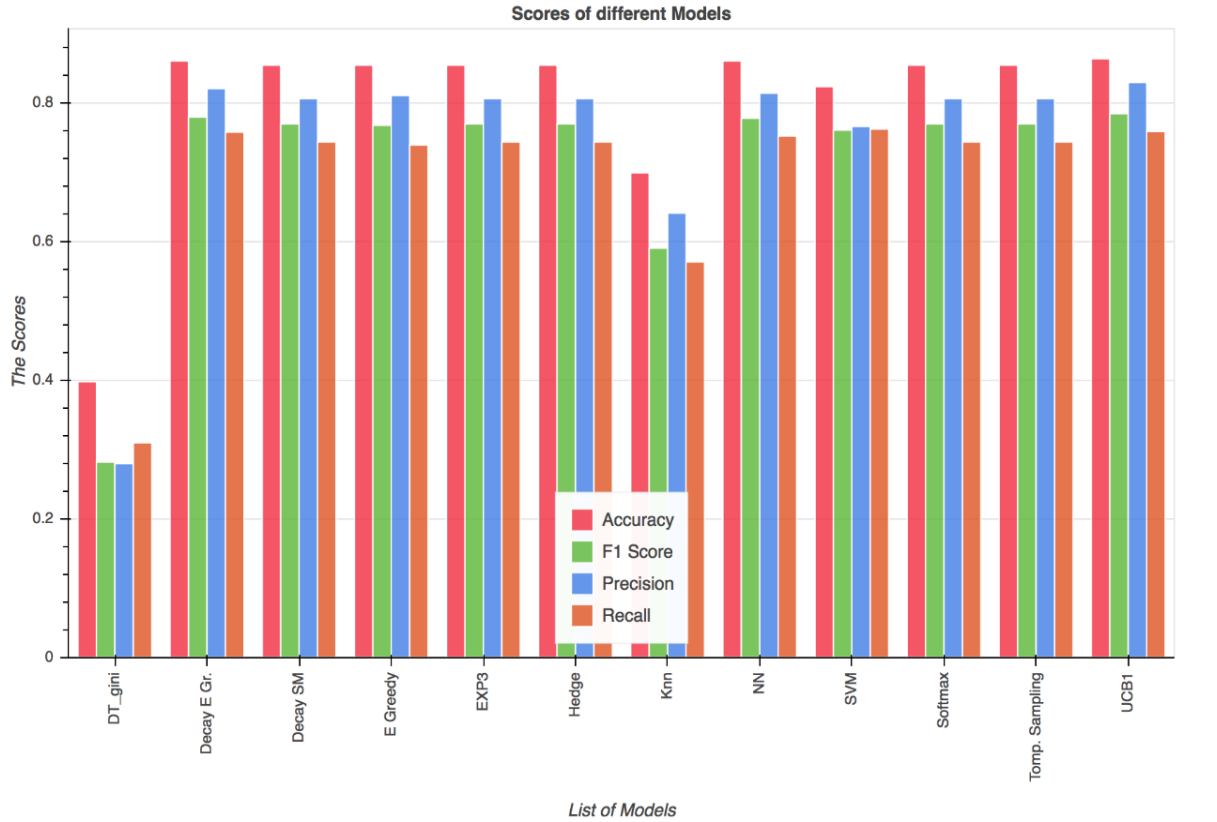
Cancer data set



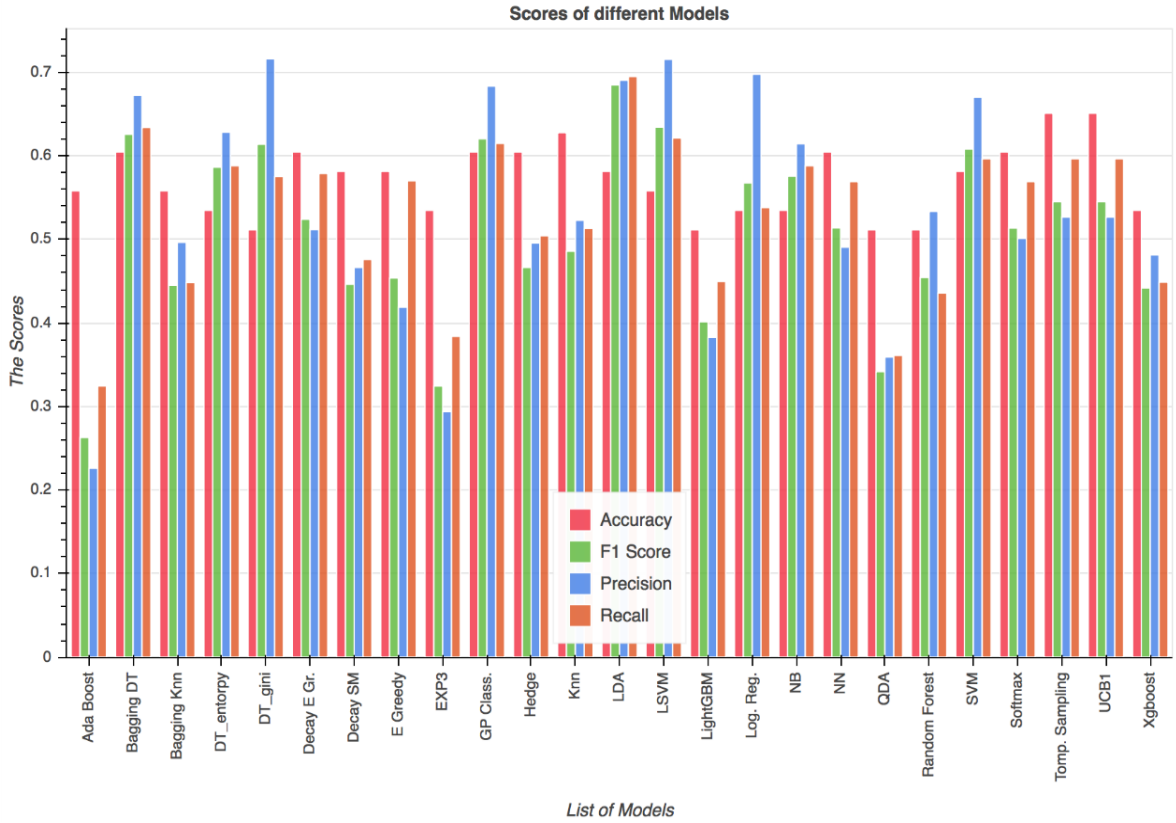
Car data set



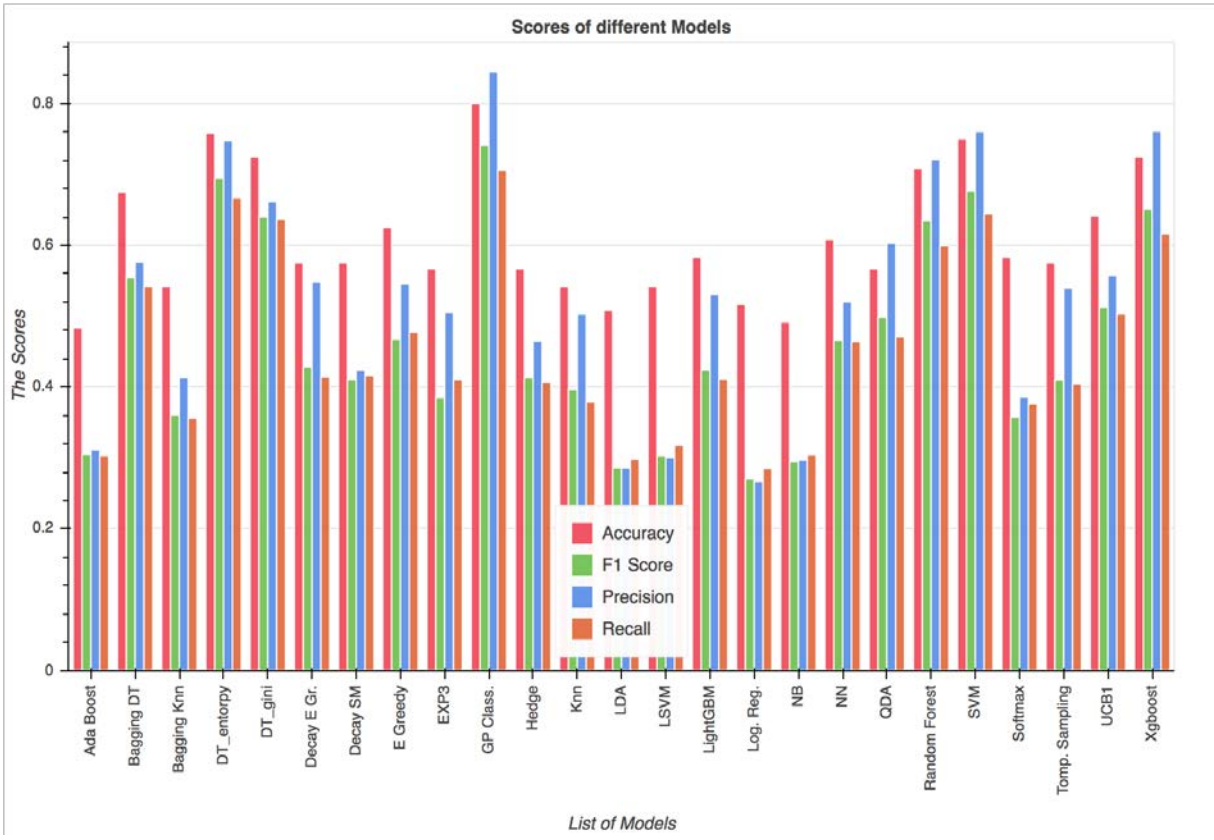
Chest data set



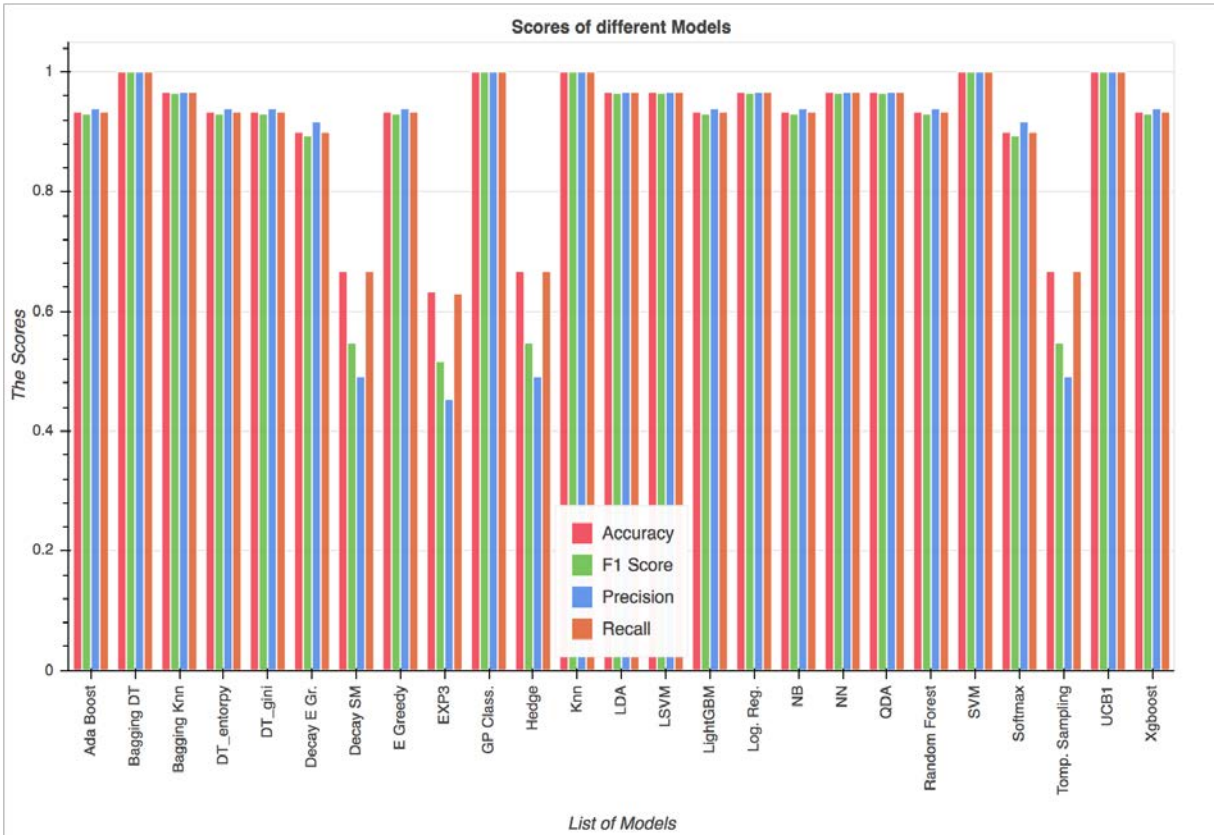
Face data set



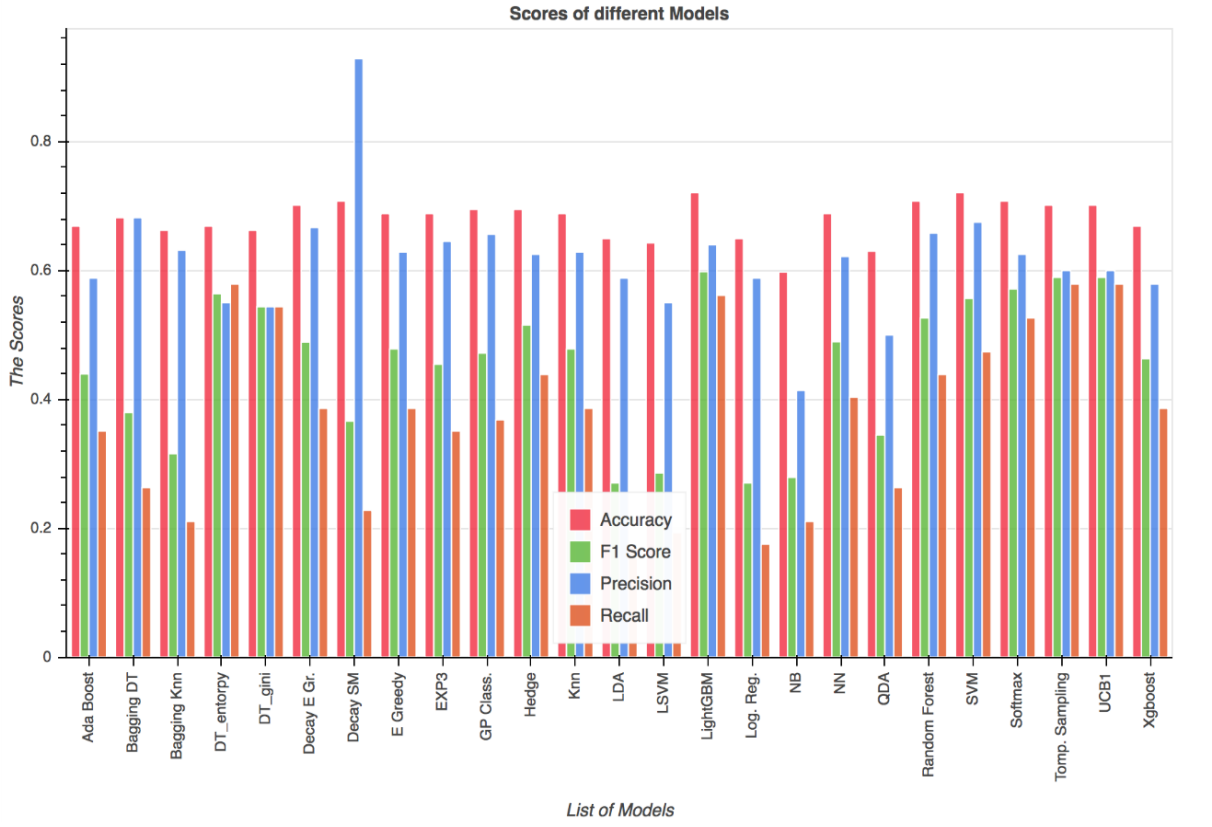
Glass data set



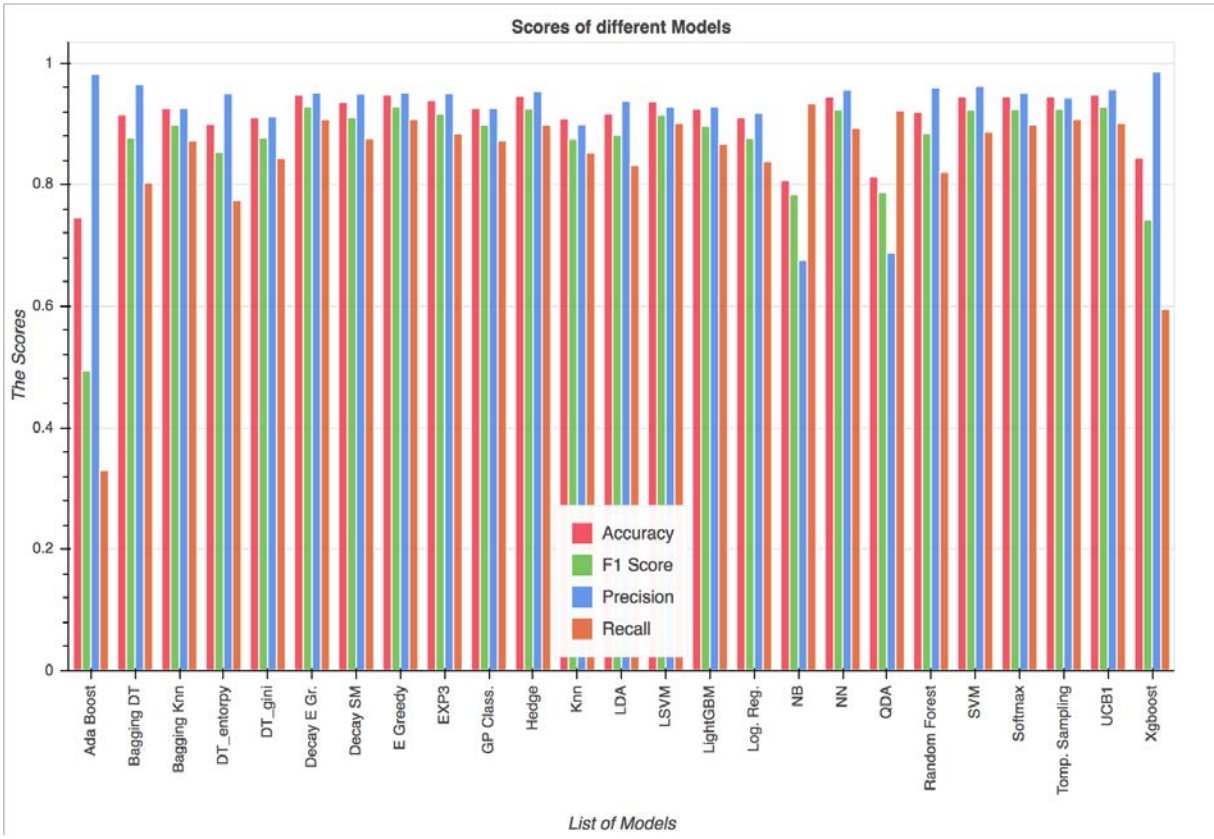
Heart data set



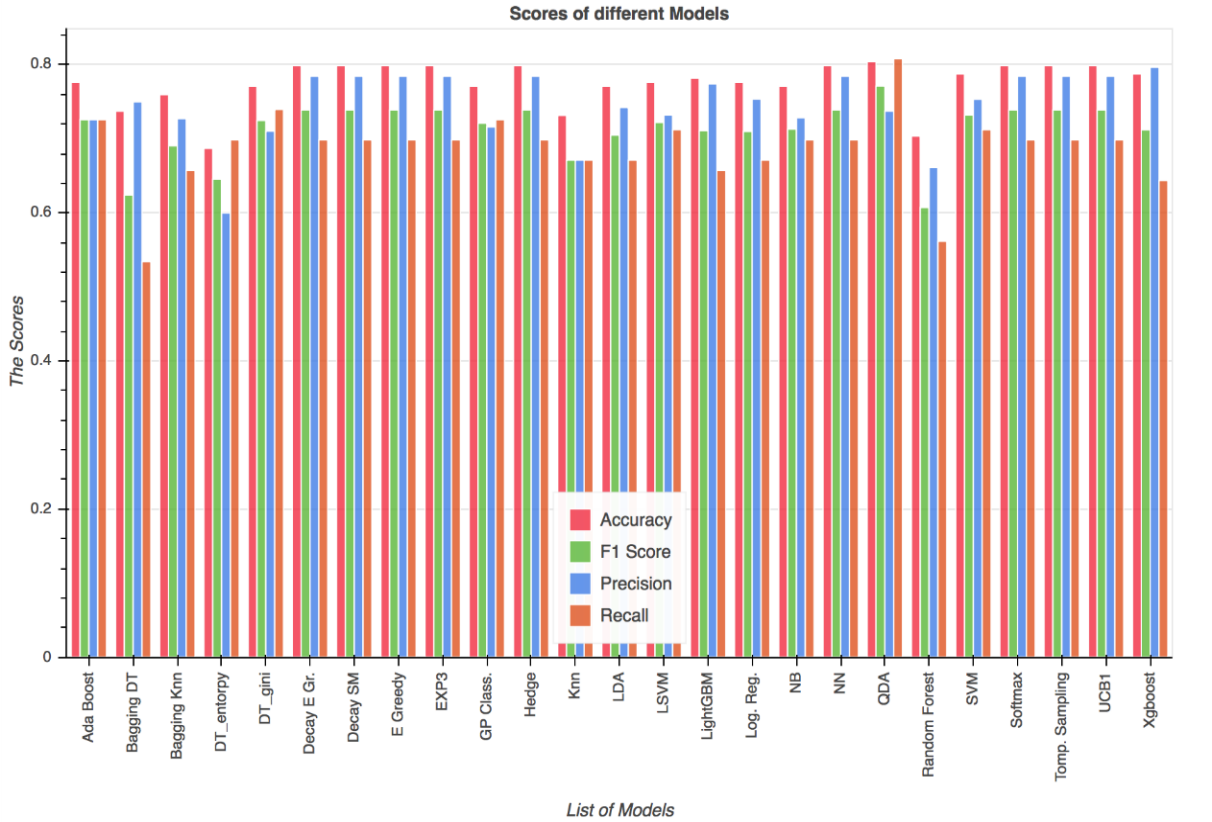
Iris data set



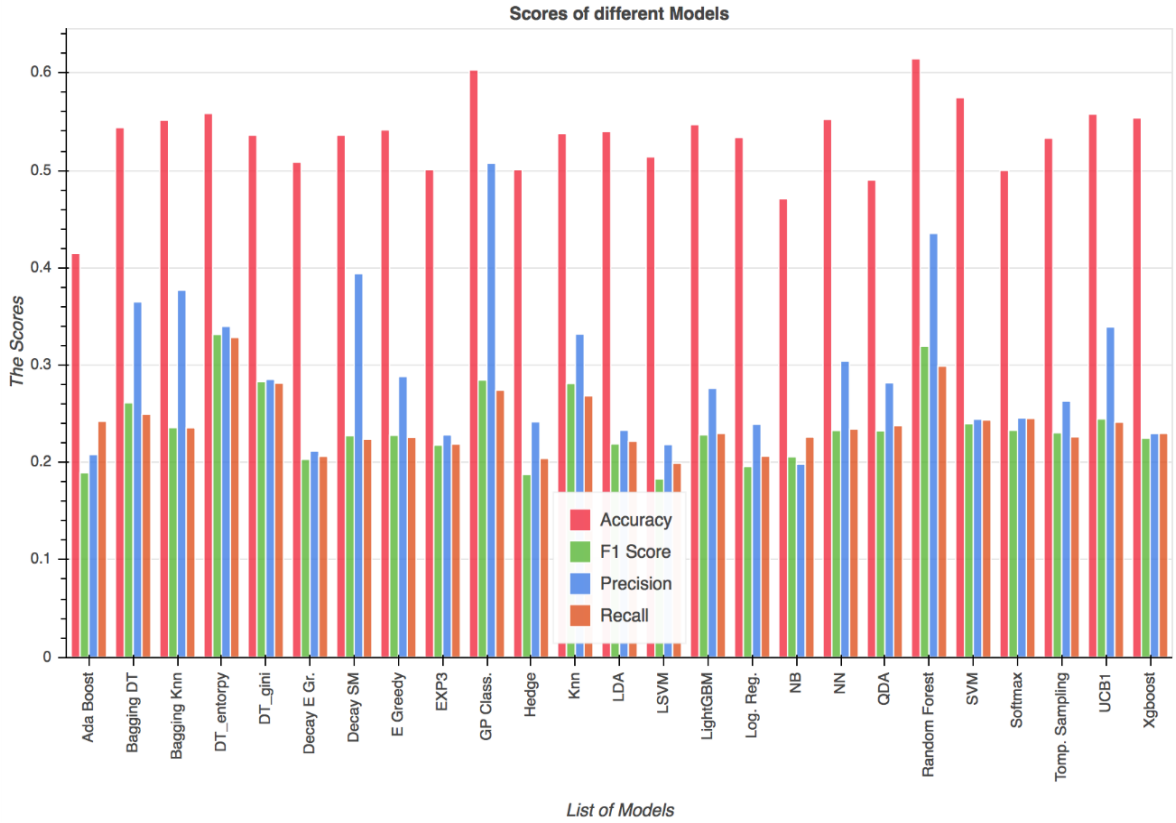
Pima data set



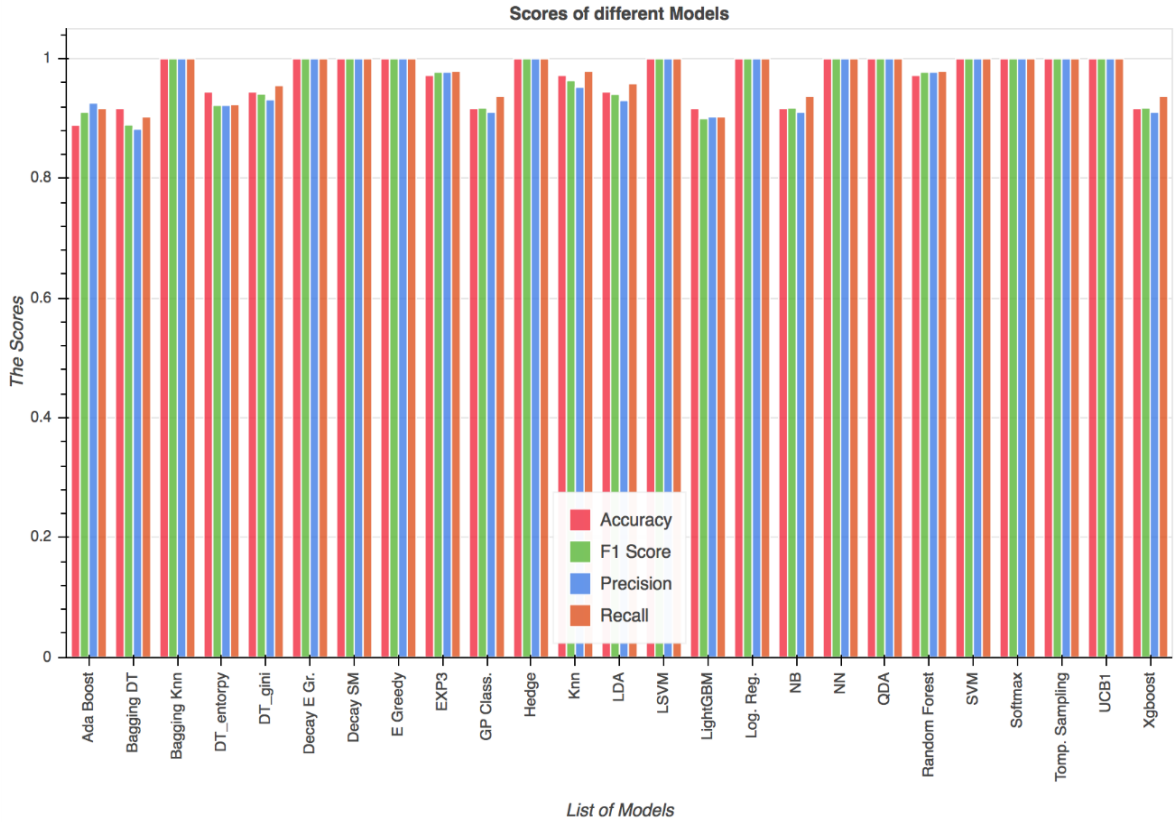
Spam data set



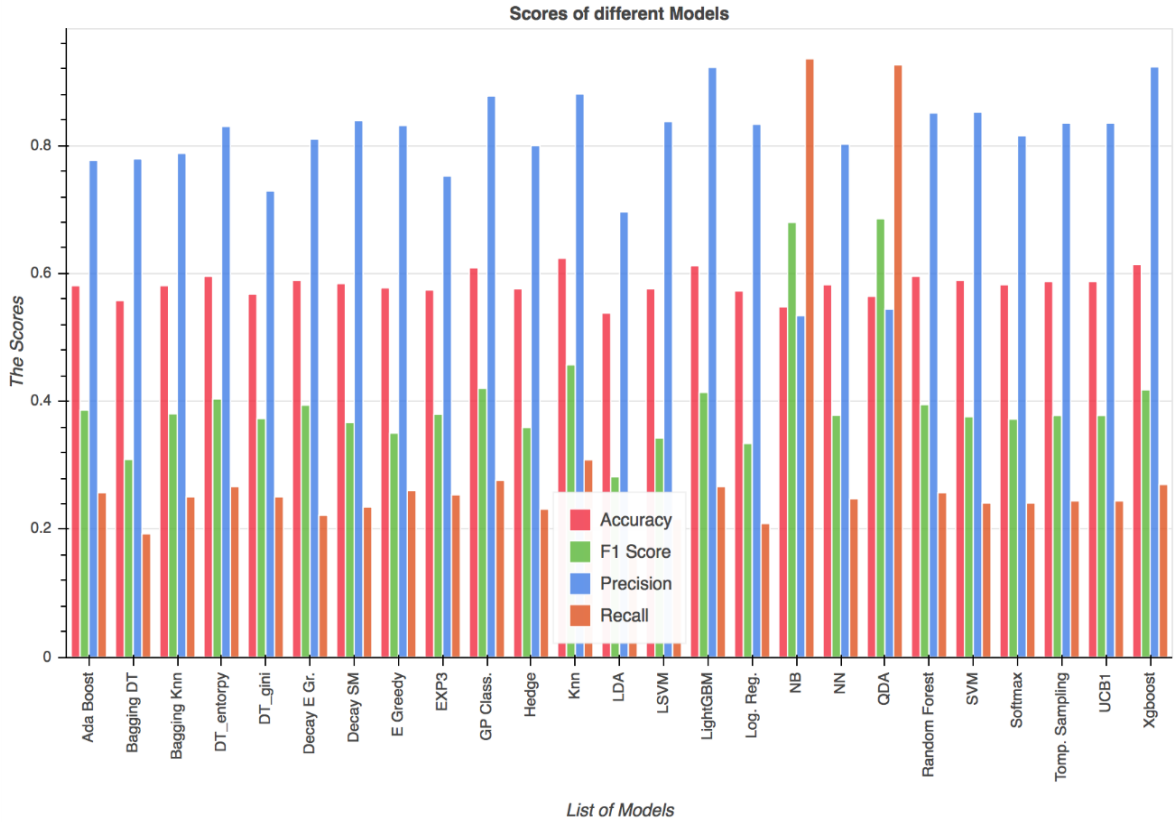
Titanic data set



Wine quality data set

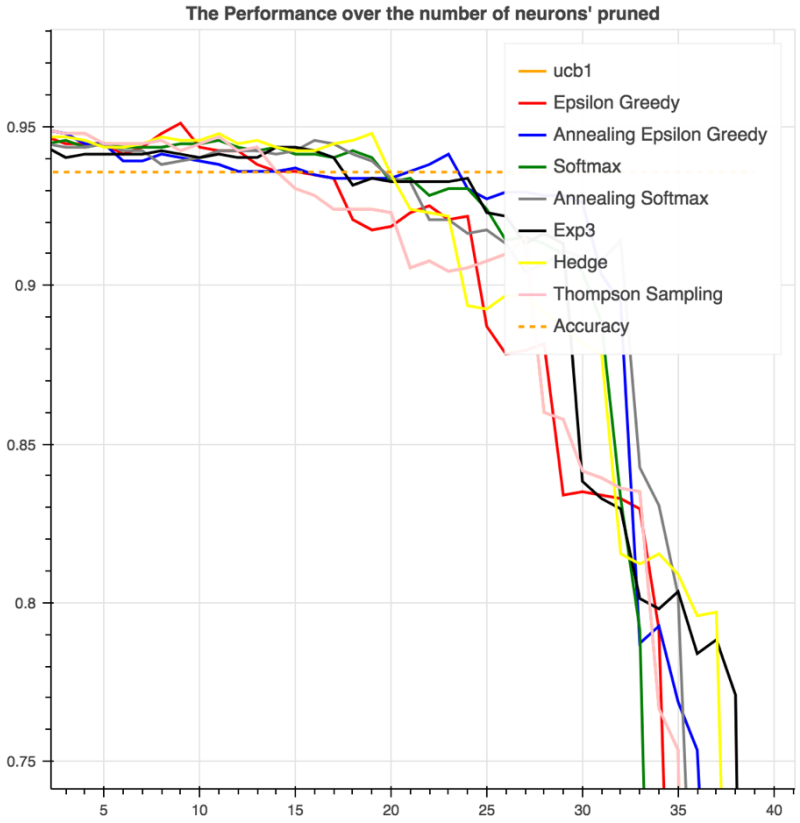


Wine data set

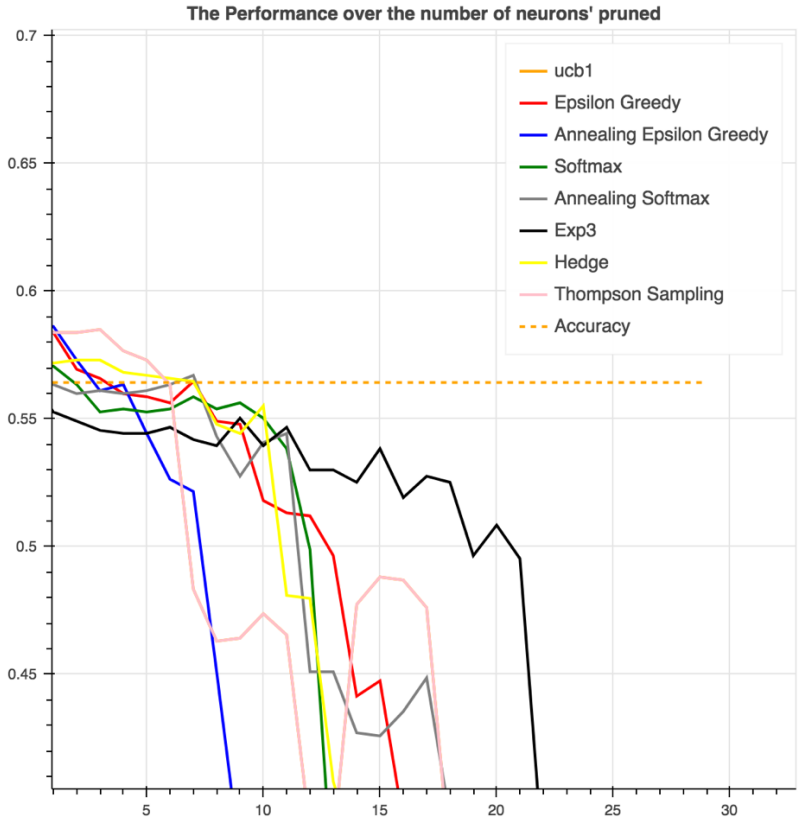


Valley data set

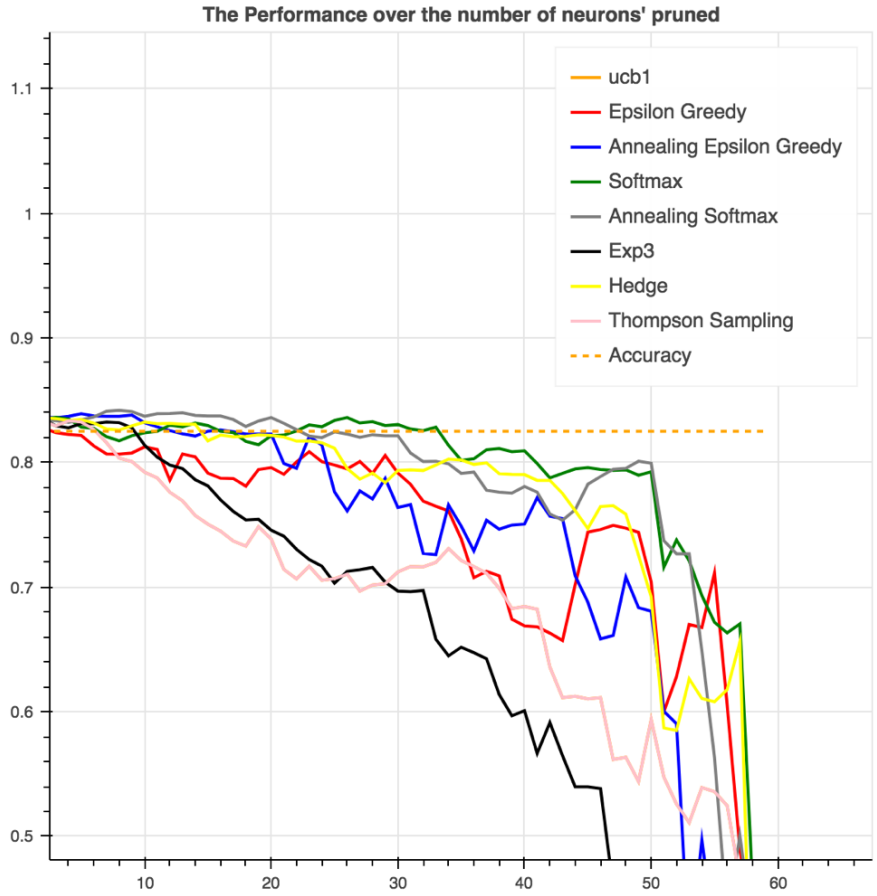
Appendix 3 Visualization on Testing Different Data sets



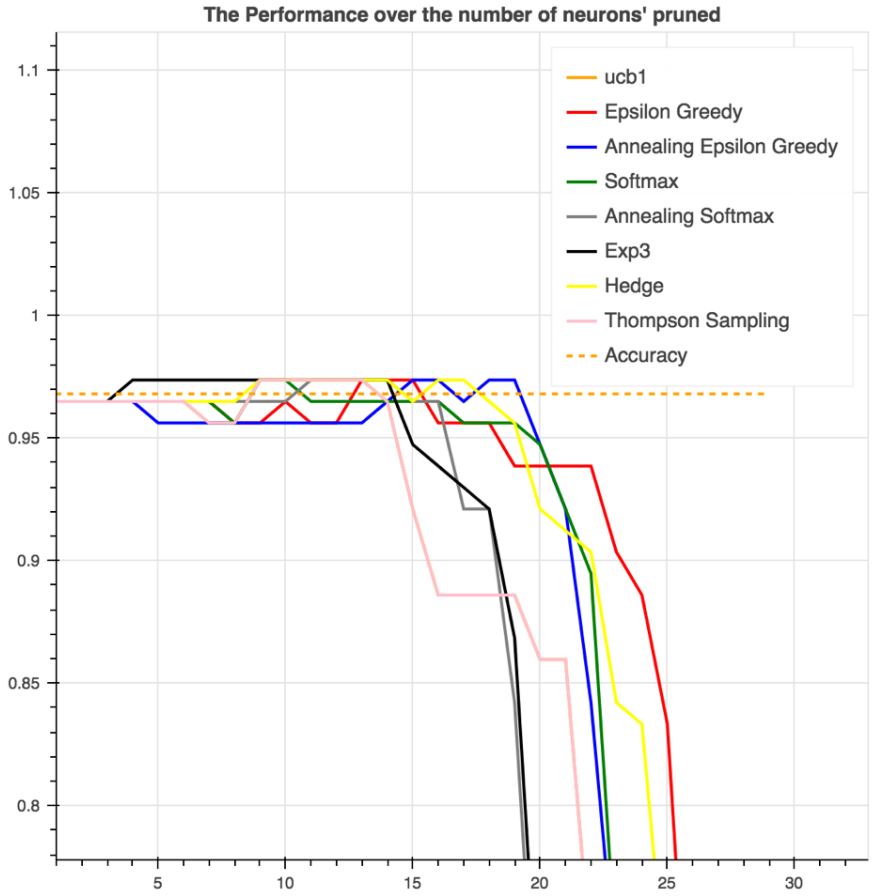
Spambase Data set



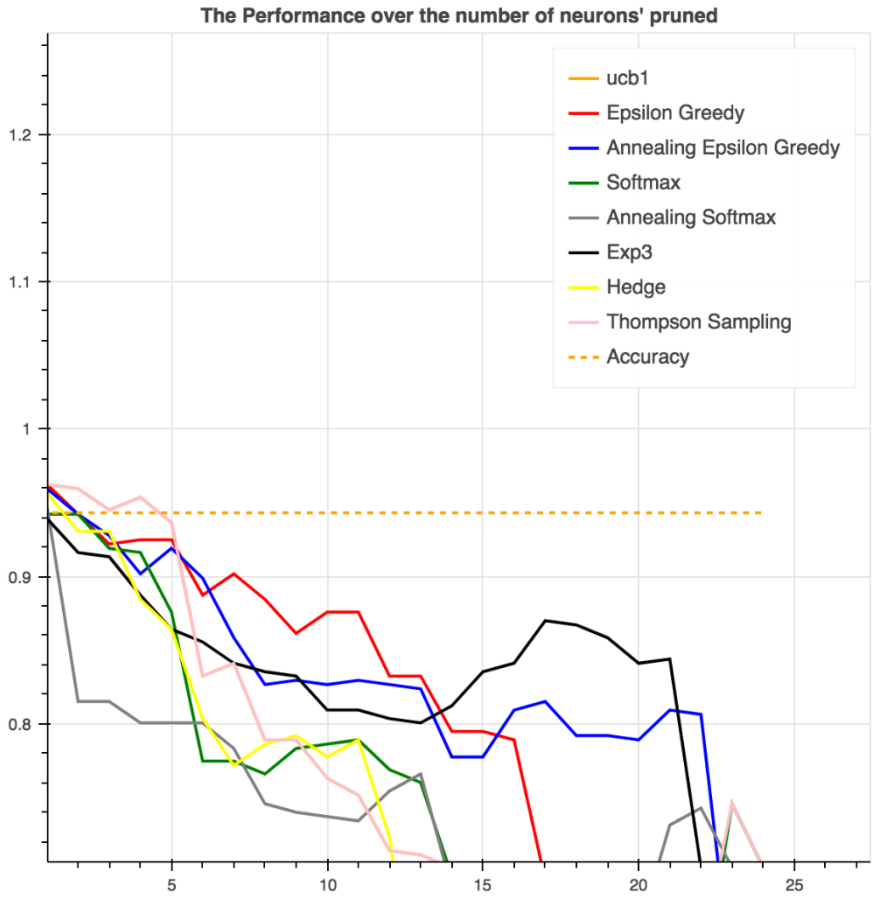
Abalone Data set



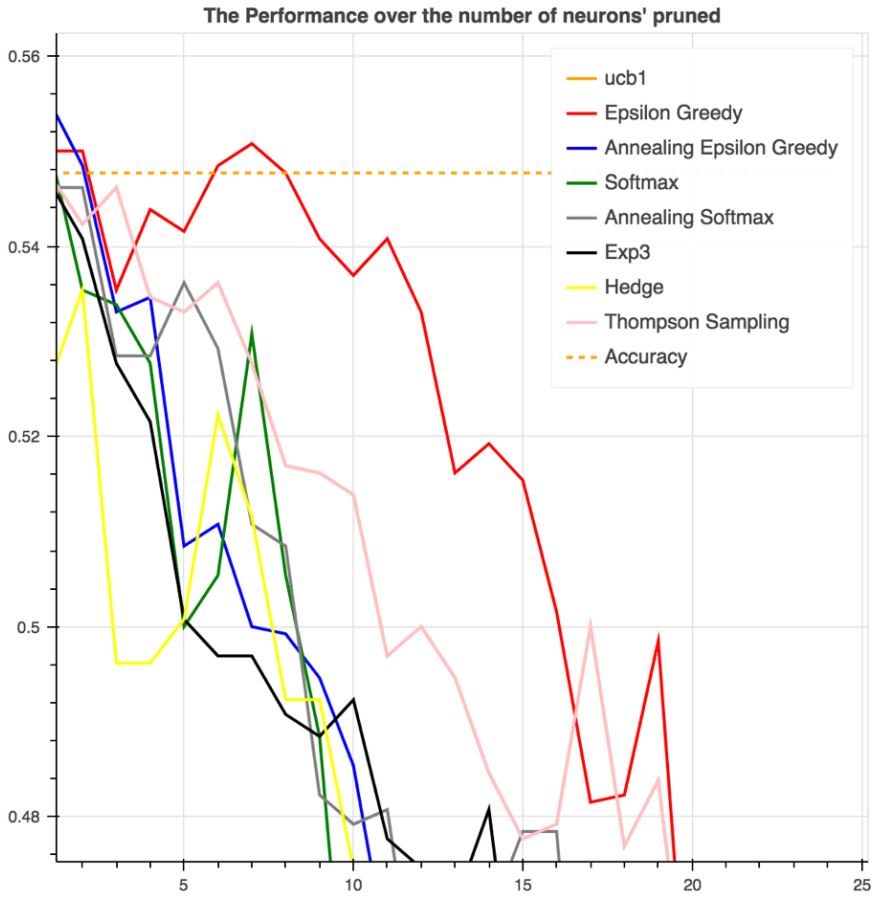
Adult Data set



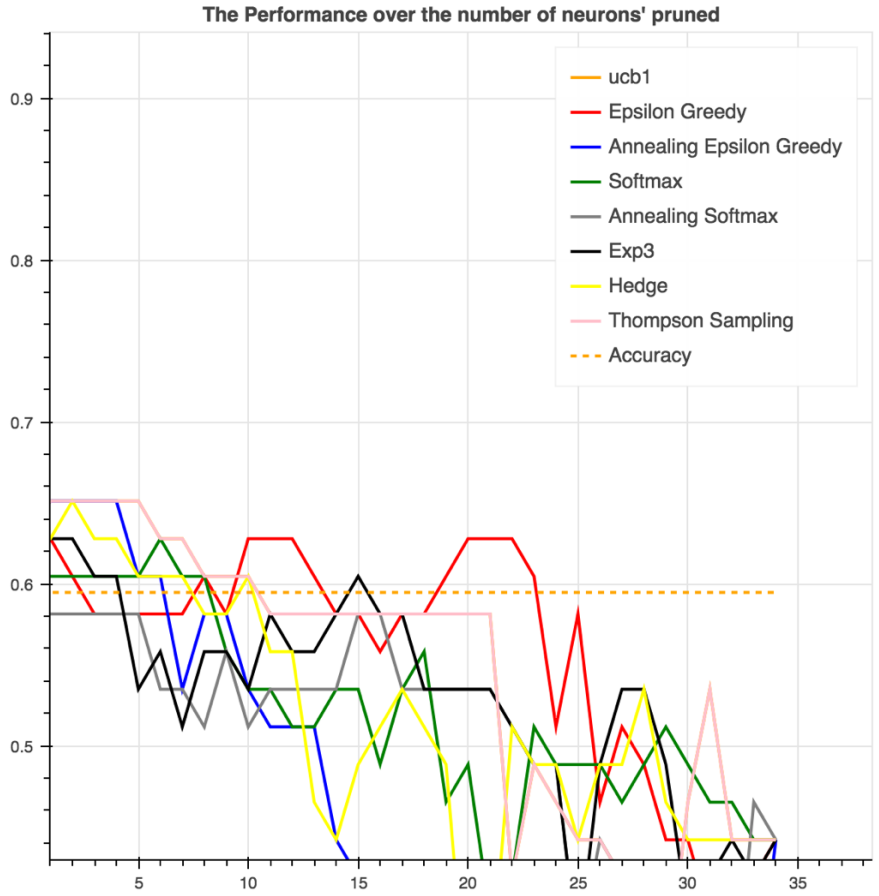
Cancer Data set



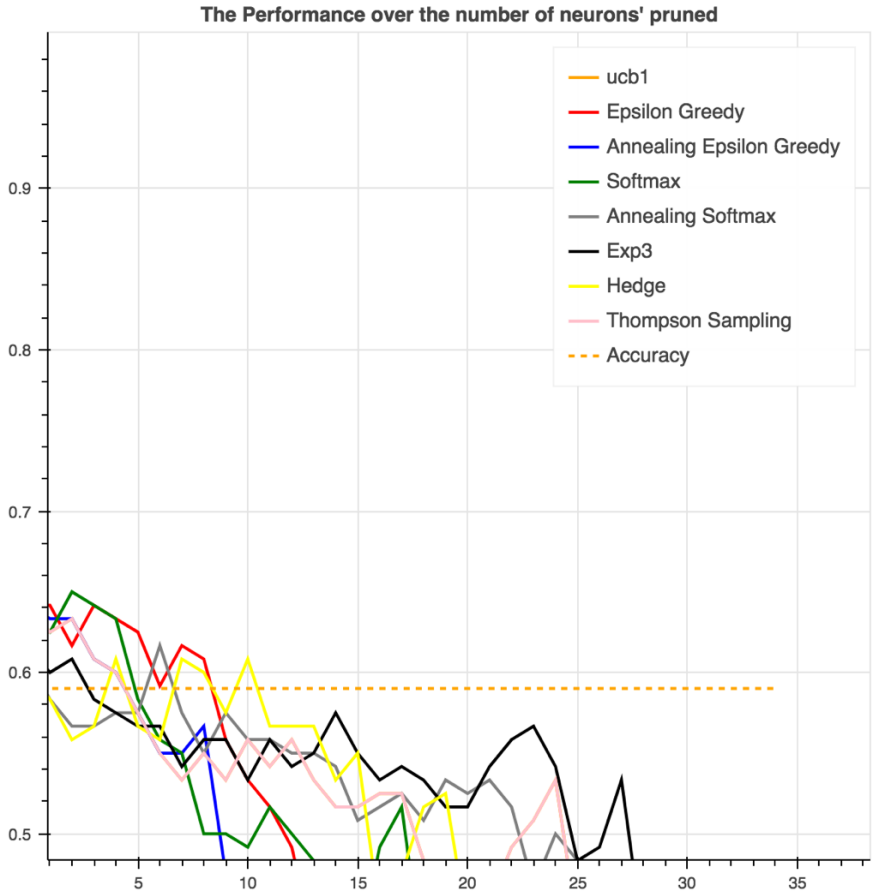
Car Data set



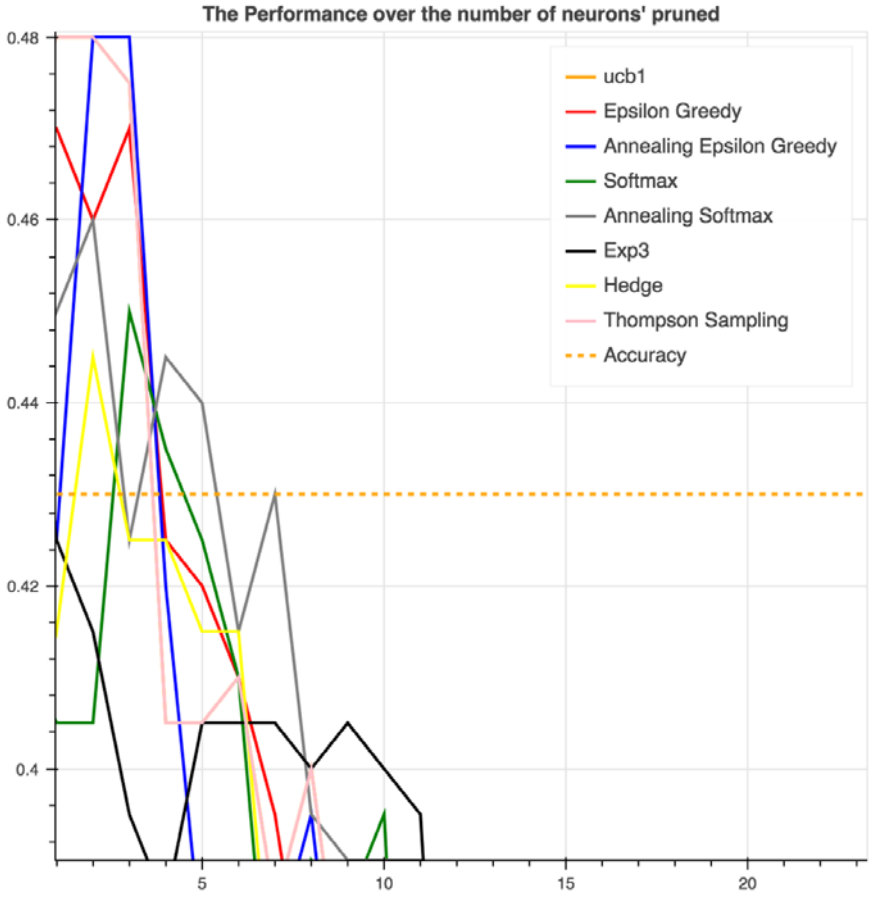
Wine Quality Data set



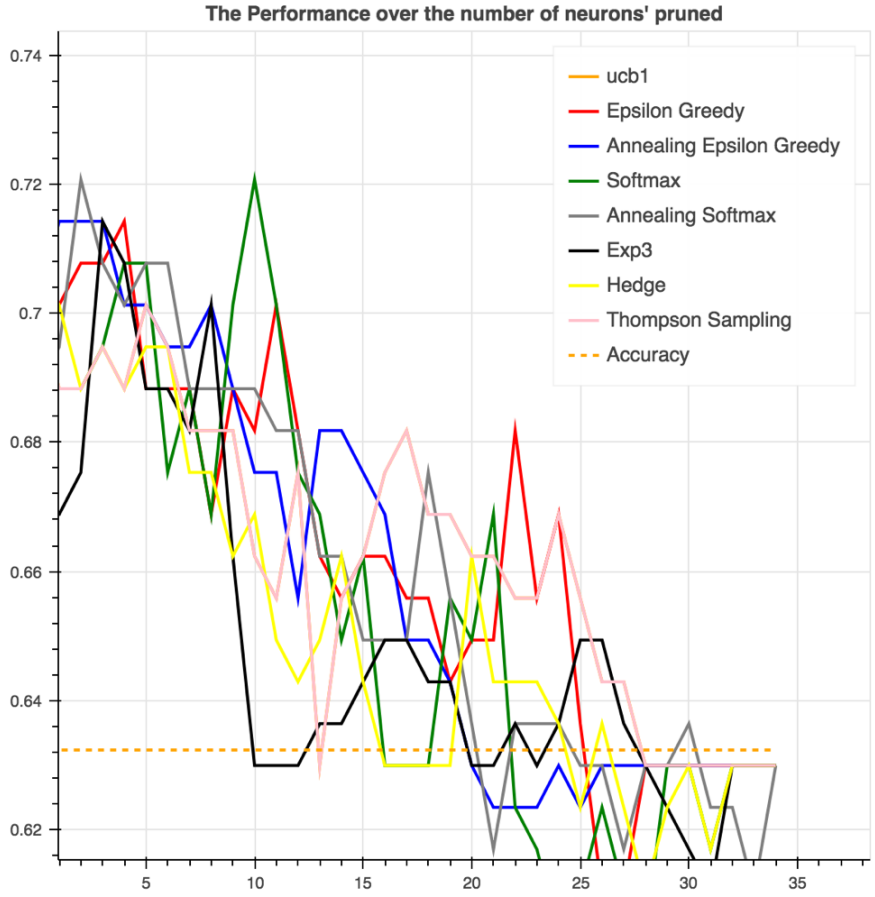
Glass Data set



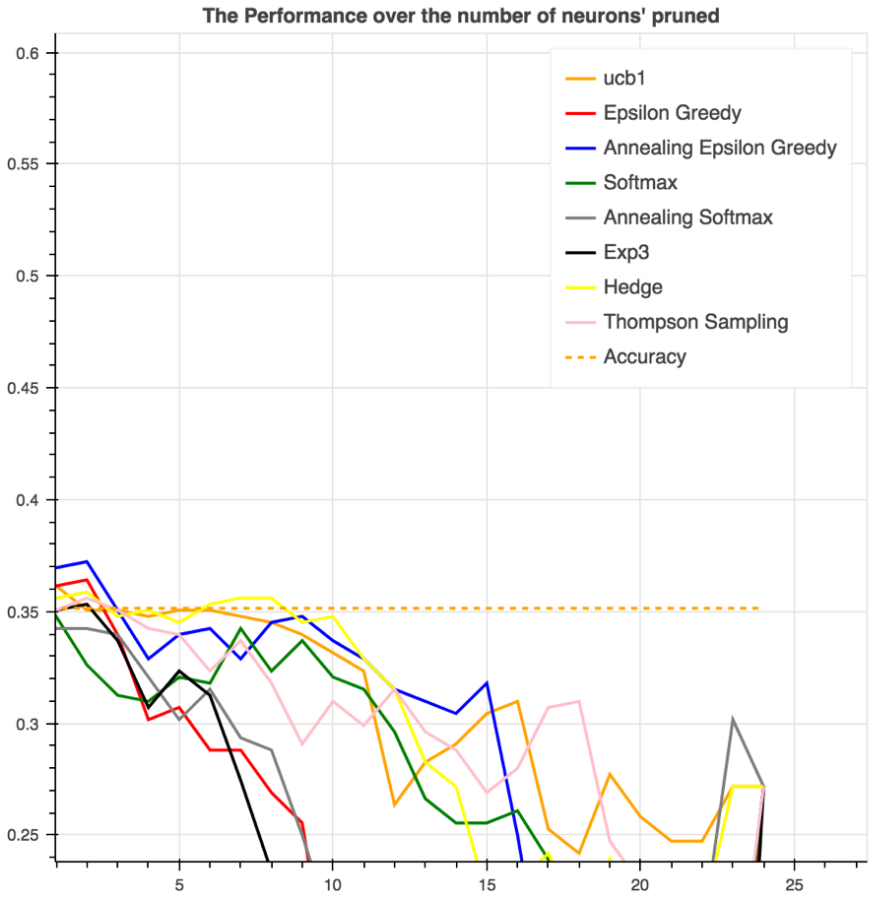
Heart Data set



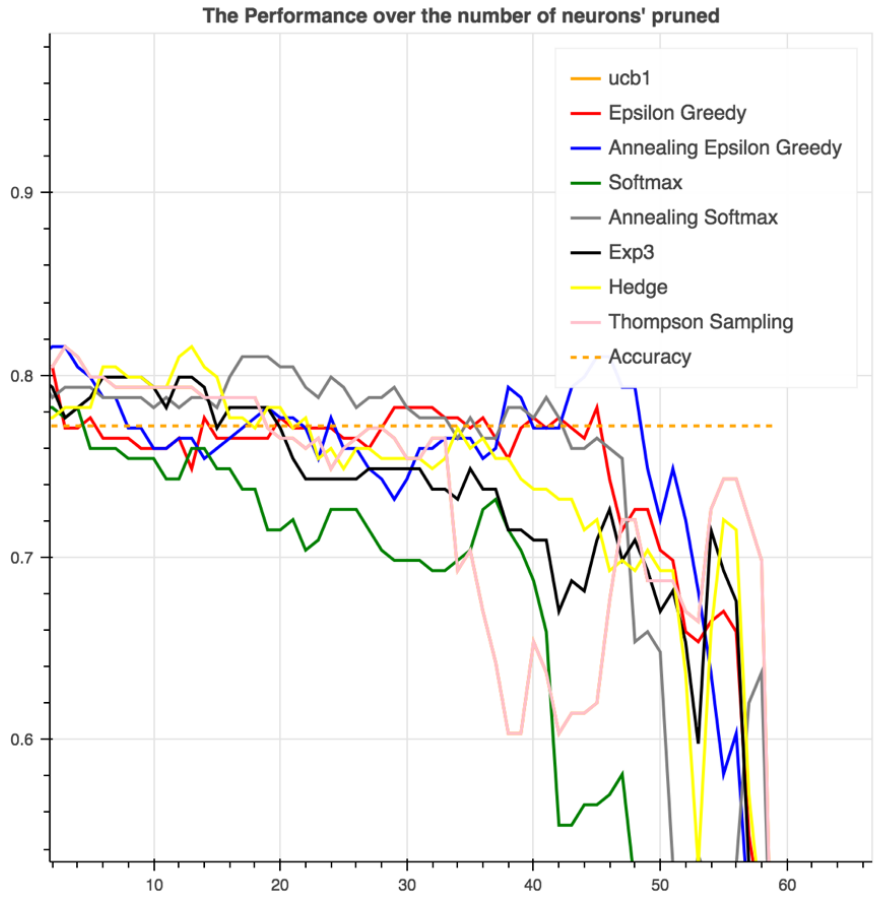
Chest Data set



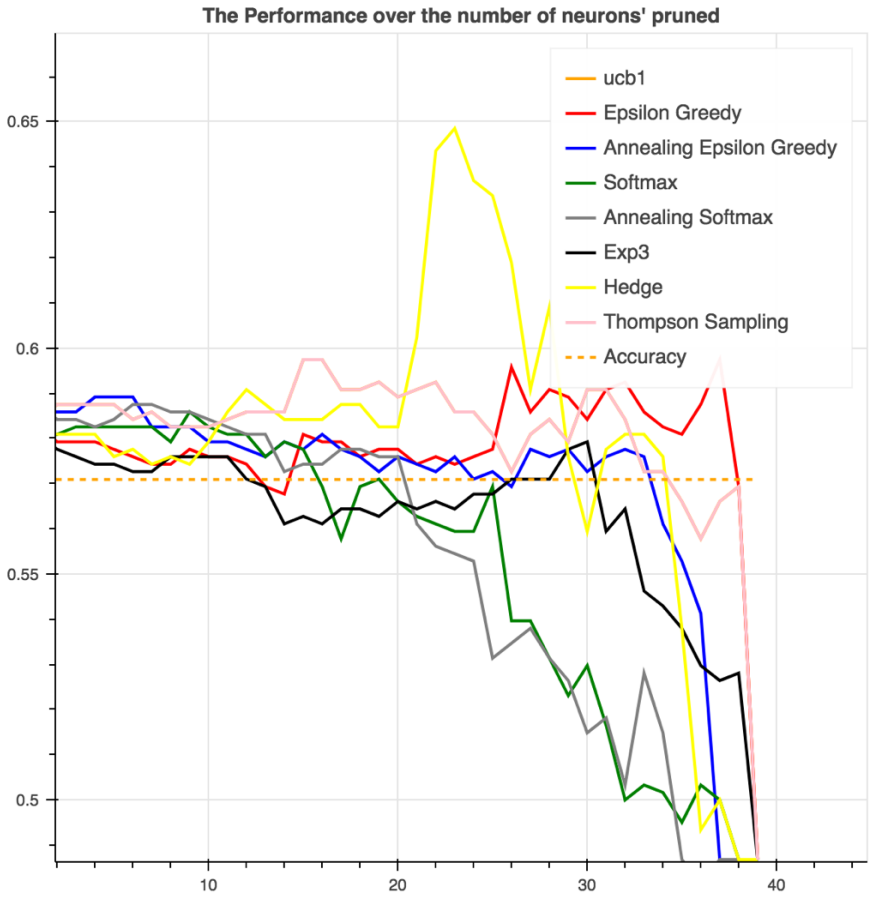
Pima Data set



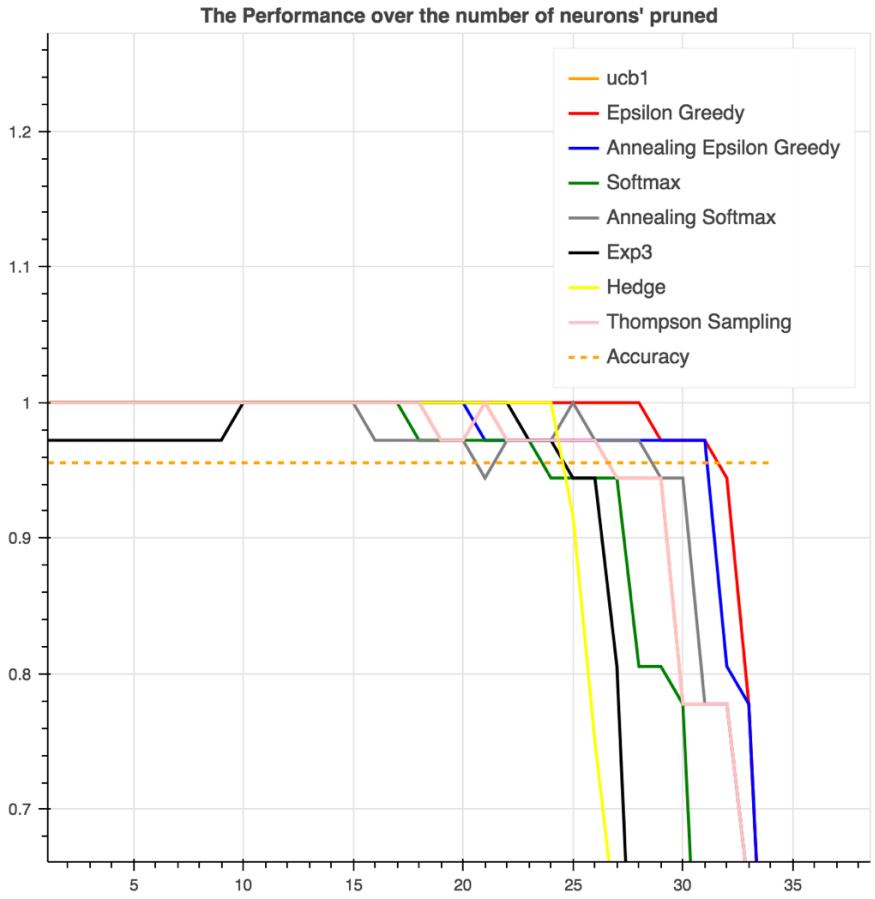
Poker Data set



Titanic Data set



Valley Data set



Wine Data set

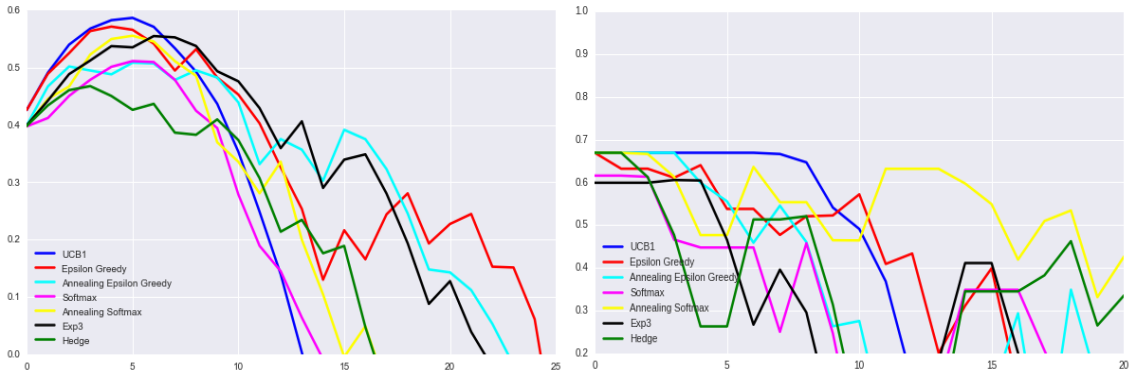
Appendix 4 Testing MAB Pruning Algorithms in Regression Data sets

Likewise, the classification data set, we compared MAB pruning algorithms between each other to see which pruning technique was the best and then we compared with the other classifiers. The comparison includes the original neural network models and the other machine learning models. In addition, the code of the implementation and complete results are available online[74].

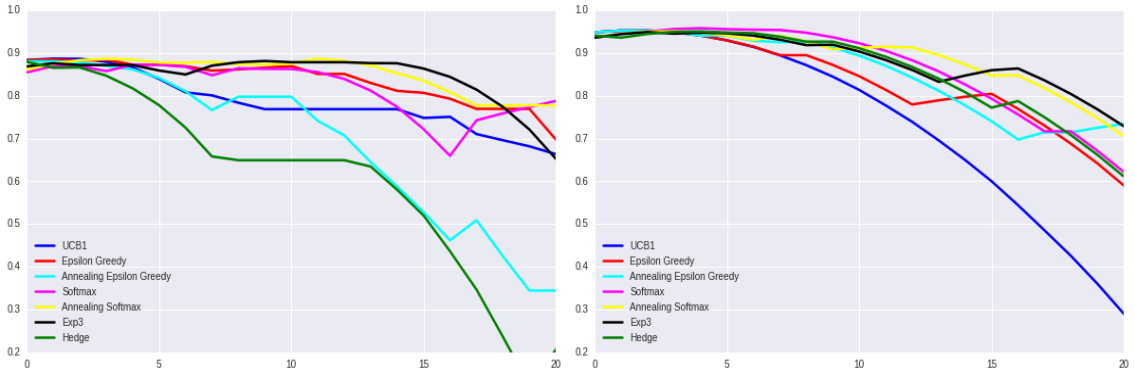
Comparing the results of MAB pruning algorithms among each other and with the original unpruned networks:

R-squared is used to test the performance between the models in the regression data sets. Figure 11.10 shows the r-squared between different MAB algorithms over a number of pruned neurons. In Figure 11.10, the behaviour of all MAB pruning algorithms is nearly the same and in general UCB1 is mostly more stable than the others.

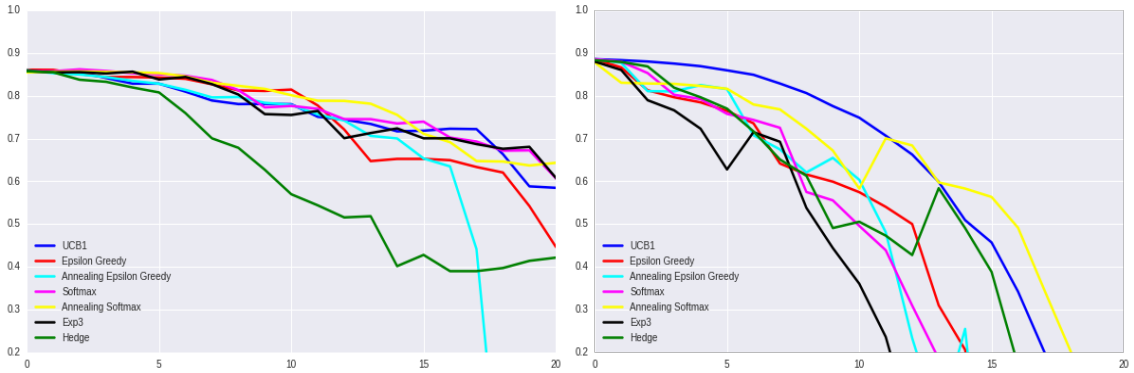
[74] https://github.com/SalemAmeen/regression



Boston House data set Airfoil Self-Noise Data Set



Auto MPG Data Set Computer Hardware Data Set



Concrete Compressive Strength Parkinsons Telemonitoring Data Set

Figure 11.10: R-squared on different MAB pruned models on x-axis shows the number of pruned neurons and in y-axis shows the accuracy.

Figure 11.11 shows that all proposed algorithms compared to the original networks when 20% of the original unpruned networks are pruned. Although these networks are small,

pruning them using MAB (specially algorithms based on UCB1 and Thompson Sampling) leads to good new models and sometimes better than the original.

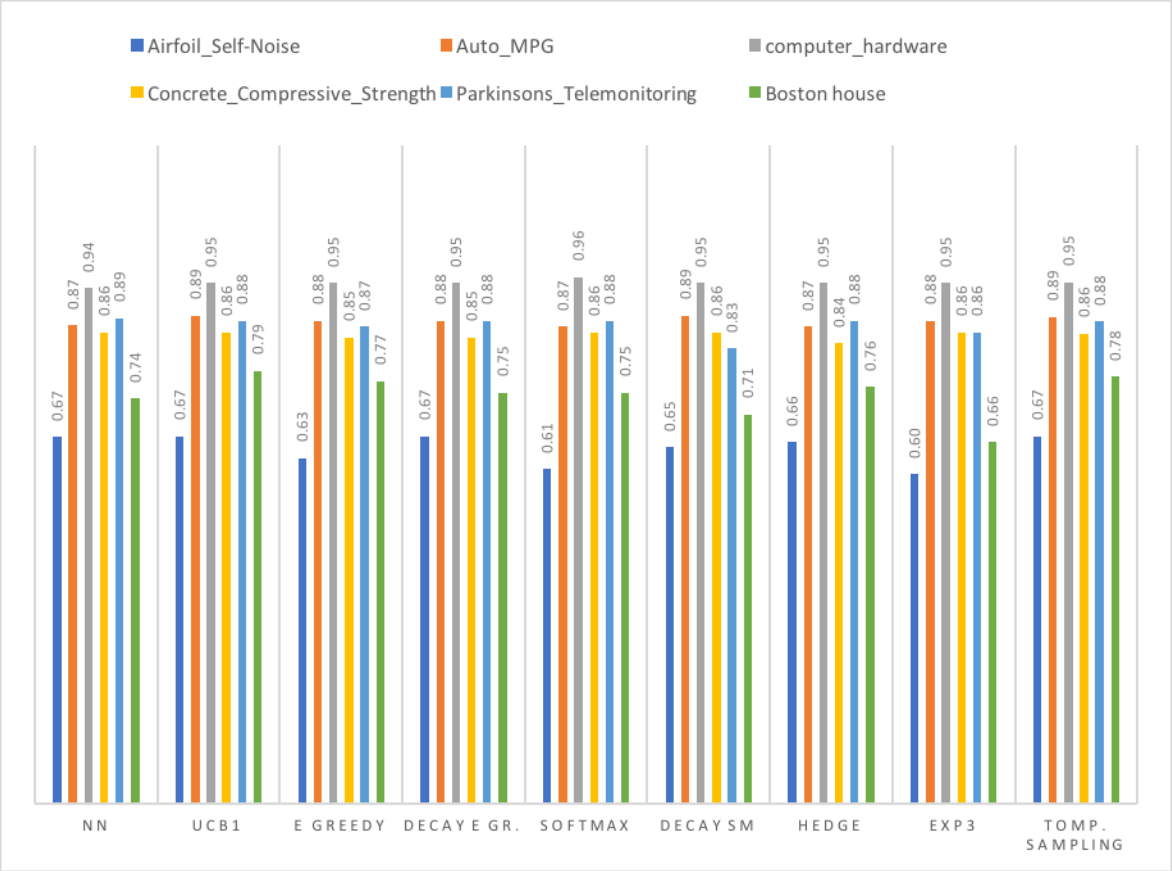


Figure 11.11: R-squared of MAB algorithms and the original unpruned model tested on regression testing data sets

Comparing MAB pruning algorithms with other models:

We compare proposed pruned models with the other regression models, as shown in Figure 11.12. From the Figure 11.12, we can see that the performance has been improved when the models are pruned. In many cases, the UCB1pruning algorithm achieves the best results among the all models.

The p value of applying Friedman test on the r squared to these six data sets and 19 different algorithms is 7.45×10^{-07}, which is less than 0.05. It indicates that there is significant difference between the mean of the different algorithms. Table 11-1 shows the average

difference between the algorithms where the higher numbers have better results based on the r squared.

Table 11-1 shows proposed algorithms based on UCB1, Hedge, Softmax and Decay Epsilon-Greedy pruned the original model and the results are improved even by pruning nearly 20% of the original models. The next step is to prove whether this improvement is significantly different or not by using Nemenyi test between all the algorithms and the results as shown in the Figure 11.13.

Figure 11.13 shows that there is a large improvement when using UCB1. Pruning based on UCB1 made the neural networks model even better than linear regression (Ordi. Least Sq), LASSO, Bayesian Ridge and SVM. Hedge pruning algorithm improves the original model and makes it statistically better than the regression model and LASSO. Softmax and Decay Epsilon-Greedy pruning algorithm statistically improved the original unpruned model to get better results than SVM regression.

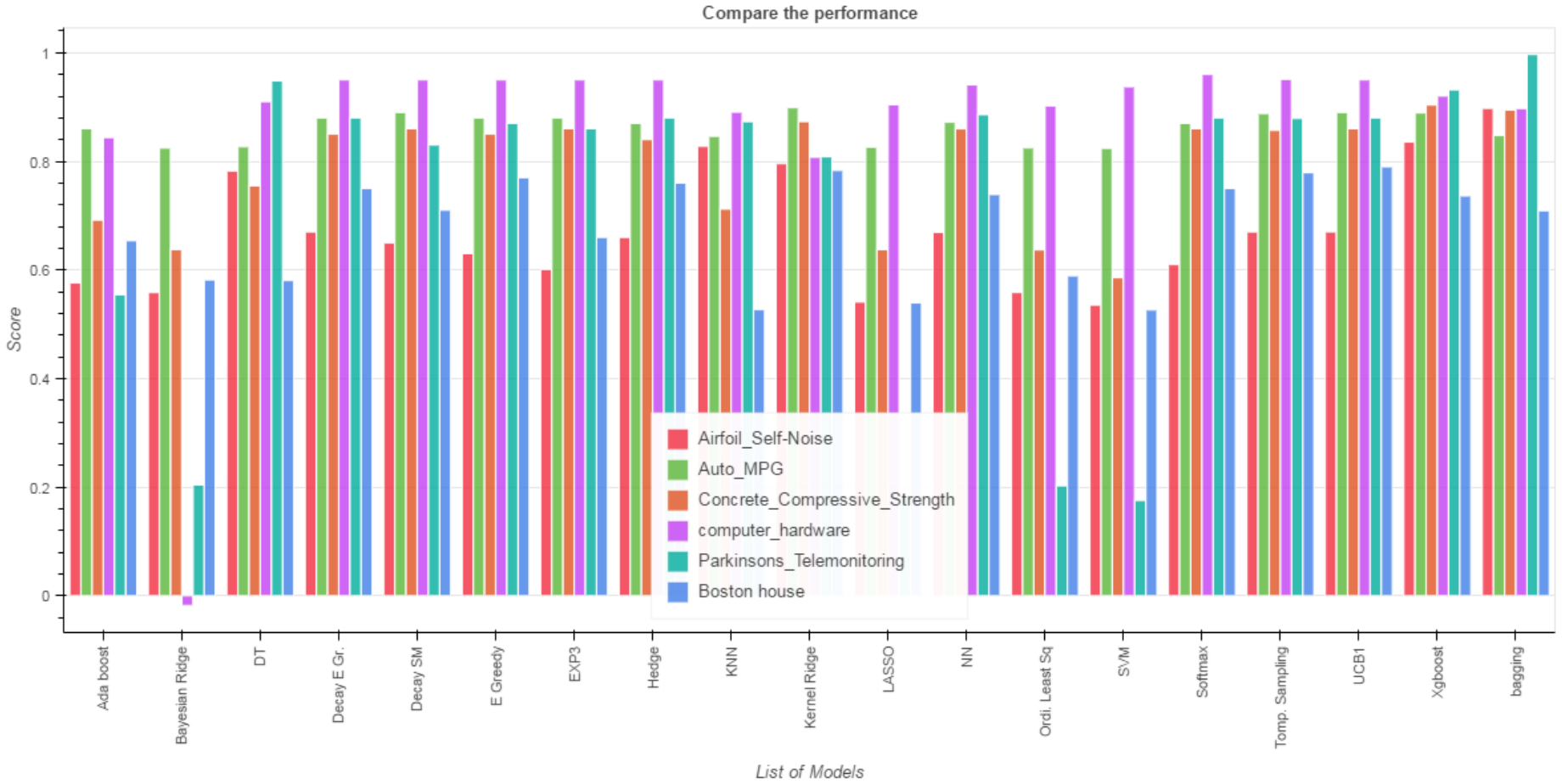


Figure 11.12: R-squared between MAB pruning algorithms pruned 25% of the original model and other regression models

Name of Method	Mean Rank
UCB1	15.500
Xgboost	15.0
Hedge	14.083
Kernel Ridge	13.0
bagging	12.833
Softmax	12.833
Decay E Gr.	12.750
NN	12.167
Decay SM	12.083
EXP3	11.750
E Greedy	11.667
Tomp. Sampling	10.667
DT	9.500
KNN	7.500
Ada boost	5.500
Ordi. Least Sq	4.0
LASSO	3.500
Bayesian Ridge	3.167
SVM	2.500

Table 11-1: Results of ranked R squared results based on Nemenyi test, which is used to compare the different models on six different data sets.

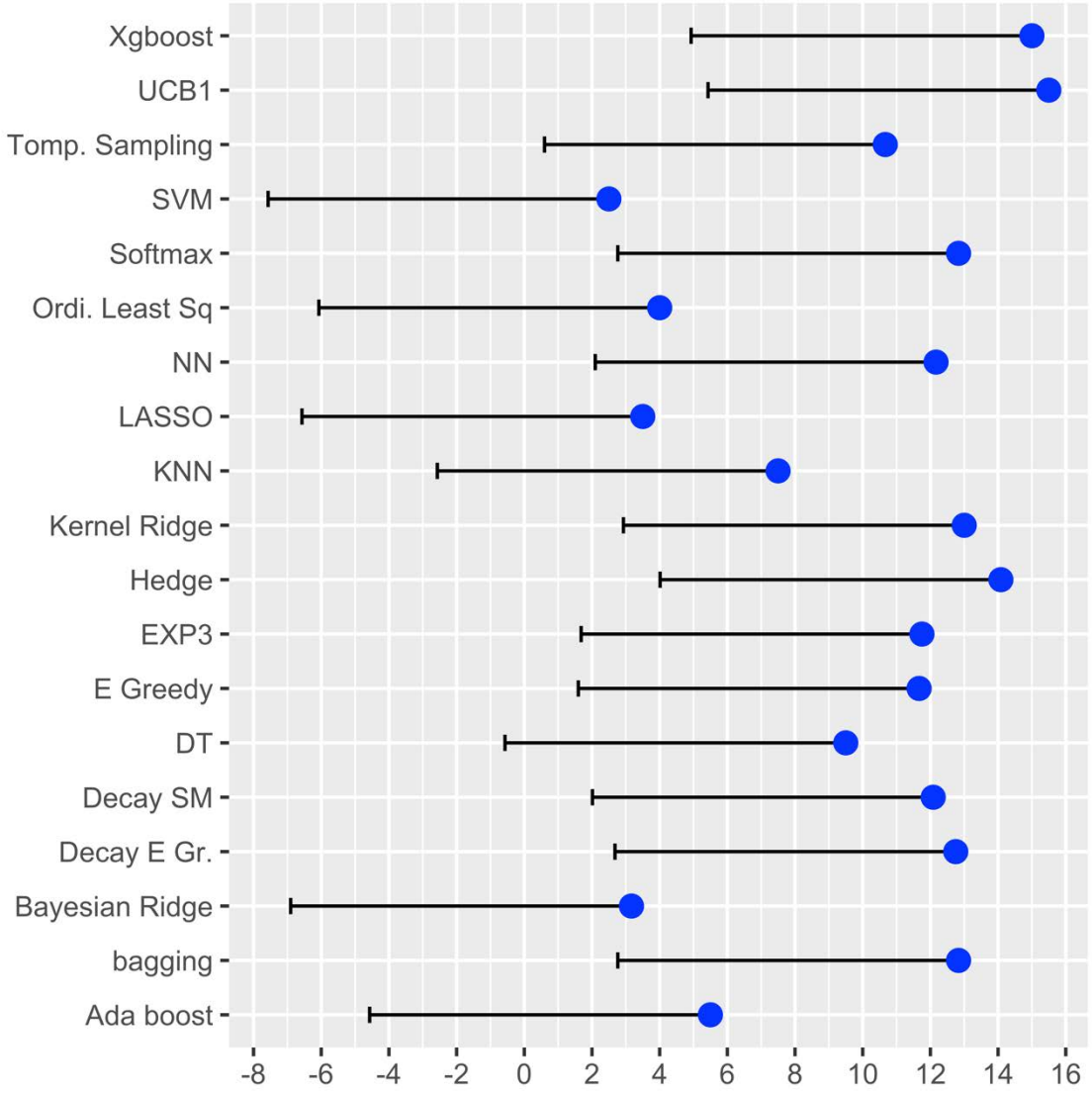


Figure 11.13: Comparison of all classifiers against each other with the Nemenyi test. Horizontal lines show the critical difference away from proposed pruning methods and any other methods. Groups of regressions that are not significantly different (p = 0.0.5) are out of the lines from proposed methods. CD=10.07