AUTOMATED LEARNING RATE SEARCH USING BATCH-LEVEL CROSS-VALIDATION

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

DUYGU KABAKCI

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
COMPUTER ENGINEERING

JULY 2019

Approval of the thesis:

AUTOMATED LEARNING RATE SEARCH USING BATCH-LEVEL CROSS-VALIDATION

submitted by **DUYGU KABAKCI** in partial fulfillment of the requirements for the degree of **Master of Science in Computer Engineering Department, Middle East Technical University** by,

Prof. Dr. Halil Kalıpçılar
Dean, Graduate School of **Natural and Applied Sciences** ______________

Prof. Dr. Halit Oğuztüzün
Head of Department, **Computer Engineering** ______________

Assist. Prof. Dr. Emre Akbaş
Supervisor, **Computer Engineering, METU** ______________

Examining Committee Members:

Assoc. Prof. Dr. Sinan Kalkan
Computer Engineering, METU ______________

Assist. Prof. Dr. Emre Akbaş
Computer Engineering, METU ______________

Assist. Prof. Dr. Mehmet Tan
Computer Engineering, TOBB ETU ______________

Date:

Name, Surname:	Duygu Kabakcı
Signature :	

ABSTRACT

AUTOMATED LEARNING RATE SEARCH USING BATCH-LEVEL CROSS-VALIDATION

Kabakcı, Duygu
M.S., Department of Computer Engineering
Supervisor: Assist. Prof. Dr. Emre Akbaş

July 2019, 58 pages

Deep convolutional neural networks are being widely used in computer vision tasks, such as object recognition and detection, image segmentation and face recognition, with a variety of architectures. Deep learning researchers and practitioners have accumulated a significant amount of experience on training a wide variety of architectures on various datasets. However, given a specific network model and a dataset, obtaining the best model (i.e. the model giving the smallest test set error) while keeping the training time complexity low is still a challenging task. Hyper-parameters of deep neural networks, especially the learning rate and its (decay) schedule, highly affect the network's final performance. The general approach is to search the best learning rate and learning rate decay parameters within a cross-validation framework, a process that usually requires a significant amount of experimentation with extensive time cost. In classical cross-validation, a random part of the dataset is reserved for the evaluation of model performance on unseen data. This technique is usually run multiple times to decide learning rate settings with random validation sets. This thesis is aimed at exploring batch-level cross-validation methods as an alternative to the classical dataset-level, hence macro, CV. The advantage of micro CV methods is that the gradient computed during training is re-used to evaluate several different learning rates. We propose automated learning rate selection algorithms that are aimed to address setting the learning rate and learning rate schedule during training. Our algorithms use micro cross-validation where a random half of the current batch (of examples) is used for training and the other half is used for validation. We present comprehensive experimental results on three well-known datasets (CIFAR10, SVHN and ADIENCE) using three different network architectures: a custom CNN, ResNet and VGG.

Keywords: Deep Learning, Object Classification, Learning Rate Search, Hyper-parameter Search, Adaptive Learning Rate, Cross-Validation

ÖZ

YIĞIN SEVİYESİNDE ÇAPRAZ GEÇERLEME KULLANARAK OTOMATİK ÖĞRENME ORANI ARAMASI

Kabakcı, Duygu
Yüksek Lisans, Bilgisayar Mühendisliği Bölümü
Tez Yöneticisi: Dr. Öğr. Üyesi. Emre Akbaş

Temmuz 2019 , 58 sayfa

Derin evrişimsel sinir ağları (ESA) sahip olduğu mimari çeşitlilik ile nesne sınınflandırma ve tanıma, görüntü bölütleme ve yüz tanıma gibi bilgisayarlı görü işlemlerinde yaygın olarak kabul görmektedir. Derin öğrenme araştırmacıları ve uygulayıcıları çok çeşitli mimarileri farklı veri kümelerinde eğitmek konusunda önemli miktarda deneyim elde etmiştir. Ancak, verilen bir ağ modeli ve veri seti için, en iyi modeli (yani en küçük test seti hatasını veren model) eğitim süresi karmaşıklığını düşük tutarak elde etmek hala zor bir iştir. Derin sinir ağlarının meta parametreleri, özellikle, öğrenme oranı ve (sönümü) programı, ağın nihai performansını oldukça etkiler. Genel yaklaşım, en iyi öğrenme oranını ve öğrenme oranının sönümü parametrelerini çapraz geçerleme (ÇG) çerçevesinde aramaktır; bu, genellikle yüksek zaman maliyeti olan çok fazla deney gerektiren bir süreçtir. Çapraz geçerleme, eğitme veri kümesi dışındaki yeni veriler ile model performansının değerlendirilmesi için veri setinin rastgele bir bölümünü bölerek, bu bölüm üzerinde eğitim aşamasının izlendiği bir tekniktir. Bu teknik, genellikle rastgele geçerleme kümeleri ile öğrenme oranı parametrelerine karar vermek için birçok kez çalıştırılır. Bu tez, klasik veri kümesi seviyesindeki, yani makro ÇG'ye alternatif olarak yığın seviyesinde çapraz geçerleme yöntemlerini araştırmayı amaçlamaktadır. Mikro ÇG yöntemlerinin avantajı, eğitim sırasında hesaplanan eğimin birkaç farklı öğrenme oranını değerlendirmek için tekrar kullanılmasıdır. Önerdiğimiz otomatik öğrenme oranı seçme algoritmaları; eğitim sırasındaki öğrenme oranını ve öğrenme oranı programını belirlemeyi amaçlar.Algoritmalarımız, mevcut veri yığınının (örneklerin) rastgele yarısının eğitim için kullanıldığı ve diğer yarısının doğrulama için kullanıldığı mikro çapraz geçerleme kullanır. Üç farklı ağ mimarisini kullanarak (özel bir ESA, ResNet ve VGG), üç tanınmış veri setinde yapılmış (CIFAR10, SVHN ve ADIENCE) kapsamlı deneysel sonuçlar sunuyoruz.

Anahtar Kelimeler: Derin Öğrenme, Nesnelerin Sınıflandırılması, Öğrenme oranı araması, Uyarlamalı Öğrenme Oranı, Çapraz Geçerleme

To my beloved family,

ACKNOWLEDGMENTS

I would like to express my most sincere gratitude to my supervisor Dr. Emre Akbaş for his constant support and help. He provided me with help and insightful comments whenever I struggled. This work could not have been accomplished without his encouragement, guidance and vision. I am deeply glad to have the chance to work with such a kind and supportive supervisor.

I would like to thank Erşan who shared this academic journey with me included with all the ups and downs. He is the one who takes care of me and makes me happy all the stressful and tired times of this period. I could not accomplish so far without his never-ending support and love.

I would like to thank my friends for sharing their experiences and encouraging me during this study. They helped me to solve many obstacles with their support. Foremost, I am grateful to all of my friends for their friendship that makes everything possible.

I would like to thank my family to make feel their love and support behind me all the time. Especially, I am grateful to my dearest parents, Nezihe and Bahattin, for always caring and supporting my education beginning from my childhood. I am also deeply thankful to my brother Deniz who always help and encourage me to achieve better.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

CIFAR-10	Labeled Subsets of the 80 Million Tiny Images Dataset
CLR	Cyclic Learning Rates
CNN	Convolutional Neural Network
DNN	Deep Neural Network
FC	Fully-connected Layer of Neural Networks
LR	Learning Rate
MCV	Micro Cross Validation
RESNET	Residual Neural Network
SGD	Stochastic Gradient Descent
SVHN	The Street View House Numbers Dataset
VGG	Visual Geometry Group

CHAPTER 1

INTRODUCTION

Scientists and inventors have been working on understanding and building decision making systems to automate everyday tasks and decision making process. By the means of this intention today, Deep learning (DL) opens an era for autonomous cars, smart cities, robotics and medical tests using massive amount of data and powerful computational capabilities. Machine learning, the larger field that subsumes deep learning, learns from data and tries to generalize for further predictions and decisions by extracting patterns and representations. Power of deep learning comes from complex hierarchical representations that build over simpler representations. For example, image of a person is represented as edges in earlier layers while latter hidden layers extend representations to more complex object parts that finally turns to object identity. The word 'deep' refers to the hierarchical depth of models.

Computer vision is one of the areas that is directly related with these smart technologies on image classification (e.g. [1, 2, 3]), object detection (e.g. [4, 5, 6]) and face recognition (e.g. [7, 8]) or pose estimation (e.g. [9, 10]) tasks in a variety of domains from robotics or security to medical imaging. Deep learning, especially deep neural networks, achieve state of the art solutions in most of these tasks which are extremely useful for both industrial developments and academic research. These capabilities and wide usage of deep neural networks cause an interest to simplify deep neural network (DNN) training, i.e the phase of learning, since obtaining the optimal model which has the smallest generalization error is not trivial especially for those who are new to the field. Our motivation in this thesis, is to explore batch-level cross-validation methods, which we call micro cross-validation (MCV, for short), for the purpose of an automated search procedure for learning rate and its decay in deep neural networks.

Learning rate, i.e. step size, seems to be one of the most important hyper-parameters for deep neural networks that highly affects the model performance. Specifically, learning rate adjusts the magnitude of network's weight updates for minimizing the loss function between real object classes and predicted classes. If learning rate is too high, the model struggles to converge to a local minima; while too small learning rate slows down convergence, hence increases the training time. Using optimal learning rate and learning rate decay utilizes the network performance by acquiring effective capacity of DNN. The common solution is to use classical cross-validation to test model performance on unseen data for different learning rate and decay selections. A large number of experiments with different learning rate and decay are conducted to decide optimal values. Adaptive optimizers (e.g. Rmsprop[11], Adam[12]) were proposed as a solution to take burden of finding optimal learning rate. They are more robust with its automated parameter-wise decay. Adam using its default learning rate can achieve high accuracy results. However, recent research ([13, 14]) points out the convergence problem of Adam optimizer. According to these recent findings, hand tuned SGD with momentum optimizer can achieve better results than adaptive optimizers. Adaptive optimizers' convergence problem can be solved with additional learning rate tuning. In short, selecting learning rate and its schedule is still an unsolved challenge for which an automated procedure would have significant impact. Our proposed methods (using micro cross-validation) explore some ideas towards automating the search for the optimal learning rate. Following sections describe the problem and proposed methods in detail.

1.1 Problem Definition

In this thesis, we explore the following problems:

Is each update (on the weights of model) during training effective? Each gradient is calculated from a mini-batch of training examples that is assumed to be representative of the whole dataset so, each gradient can be considered informative in regular training. However, if some of the updates are not as informative as others, eliminating those updates can reduce overfitting, i.e. memorization and learning noise from training dataset. Therefore, if we could define a validation rule for updates, those

-if any- uninformative updates can be eliminated easily for the sake of generalization capability. By this way, we can achieve an optimal model with its effective capacity.

Is it possible to automate the search for the optimal learning rate and learning rate schedule using micro (i.e. batch-level) cross-validation? The optimal learning rate and learning rate schedule highly depend on network topology, dataset and optimizer. The typical method for tuning these parameters for a specific task, dataset and network is to use a costly cross-validation procedure. To simplify this process, automated learning rate and learning rate schedule searching methods are highly desirable.

To sum up, the main goal of this study is examining the mentioned problems and their effects on neural network training which can help us understand neural network training process and explore proposed solutions for these problems.

1.2 Proposed Method

Deep neural network training is a delicate and complex process which aims to obtain a generalizable model efficiently. This can be achieved by selecting a good set of hyper-parameters that requires an in-depth understanding of how hyper-parameters work in optimization. An excessive number of hyper-parameters, such as learning rate, mini-batch size, regularization parameter, weight decay constant and number of hidden units increase the complexity of sequential experiments as they all need to be tuned. Guided sequential experiments with a different selection of hyper-parameters usually require prior knowledge on neural network's convergence, loss function topology and dataset to achieve a model that has smallest generalization error. After each experiment, the subsequent selection of hyper-parameters can be determined based on these factors using prior knowledge. Even carefully designed sequential experiments conducted with knowledge based assessments come with a very high amount of computational and time cost, due to the complex nature of neural network training. In this work, we examine this complex nature of deep neural network training so that we can propose a mini-batch based gradient verification and automated learning rate schedule selection methodology. Instead of the common use of standard cross-validation, i.e. using a separate validation set to decide whether the model is overfitting or underfitting, we propose to confirm and validate every single training step so that we can eliminate uninformative and noisy training steps. To be able to validate a training step, convenience of a training step is defined such that gradient step update should be acceptable and beneficial for another random mini-batch of dataset. Also, learning rate , i.e. step size, of each training step should be selected to maximize validated informative gradient steps. Main steps of the proposed methods can be defined as follows:

- Validating each training step/gradient update using another random validation mini-batch.
- If a training step is found to be acceptable, learning rate that obtains the most decrease on validation mini-batch's loss is selected for that step.
- Preparing a complete learning rate schedule as an automated procedure using random validation batches.

Mainly, we designed experiments to answer the following questions for proposed methods:

- **[Q1]** In a setting where learning rate and its decay schedule are given, does micro cross-validation help improve generalization? Does it yield a better test set accuracy?
- **[Q2]** In what ways can we use micro cross-validation to automate the search for optimal learning rate and learning rate schedule?
- **[Q3]** Do MCV methods that automatically select learning rates improve test set accuracy over baseline methods?

In this work, we explore the answers of mentioned questions using designed experiments.

1.3 Contributions

This thesis is an exploratory first step towards batch-level cross-validation methods that try to automate the learning rate search. Specifically,

- Micro Cross-validation (MCV) methods apply well-known cross-validation to each training step in an innovative repetitive way instead of using a part of dataset for validation.
- MCV methods propose a step based gradient validation that uses decrease of loss on random validation mini-batches.
- MCV methods try to maximize the effect of informative gradient steps by arranging learning rate based on random mini-batch verification. This learning rate decision in every training step constructs a complete learning rate schedule automatically.

1.4 Thesis Outline

Chapter 2 discusses the stochastic gradient descent (SGD) algorithm and its comparison with adaptive gradient descent algorithms as a background. Additionally, techniques to achieve good generalization and finding optimal learning rate problems of neural network training are explained with state of the art solutions in literature.

Chapter 3 describes the proposed methods, training step verification using micro cross-validation and automated learning rate search using micro cross-validation, which are executed as a variation of SGD with momentum and adam optimizers. Theoretical and algorithmic details of the proposed methods are also presented in this chapter.

In Chapter 4, the performance of MCV methods is presented by providing experimental results on several deep neural network architectures over popular benchmark datasets. Convolutional and residual deep neural network architectures are demonstrated with experimental results in this chapter.

Chapter 5 provides a brief summary and discussion of this work.

CHAPTER 2

BACKGROUND AND RELATED WORK

In this chapter, stochastic gradient descent (SGD) methods and adaptive optimizers are explained. Also, advances of accelerated SGD forms compared to adaptive optimizers are discussed. Cross-validation, which is a well known technique for assessment of DNN model, is explained as a basis of our proposed methods. Additionally, we define finding optimal learning rate problem and discuss impacts of using optimal learning rate on neural network training. Hyper-parameter tuning methods and various automated learning rate search methods in literature are also provided in this chapter.

2.1 Gradient Descent Optimization Algorithms

2.1.1 Stochastic Gradient Descent with Momentum

SGD and its variants are probably the most used optimization algorithms for machine learning, especially for deep learning tasks. However, It can be slow without acceleration method such as momentum [15]. When loss curvature is high and gradients are consistent but small, momentum acceleration method is designed to accumulate the gradients using moving average of past gradients. Momentum method introduce a velocity term that helps to accelerate consistent gradient direction. Momentum coefficient $\alpha \in (0, 1]$ adjusts how previous gradient contributions affect the current momentum step. 0.5 and 0.9 are typical values of momentum coefficient.
SGD with momentum update is defined in Algorithm 1:

Algorithm 1 Stochastic Gradient Descent with Momentum

```
Require: Learning rate $\epsilon$, momentum parameter $\alpha$
Require: Initial parameter $\theta$, initial velocity $\vartheta$
  while training stopping criterion not met do
      Sample a mini-batch of m examples from training set: $x_1, x_2 \ldots x_m$ with
      corresponding labels $y_1, y_2 \ldots y_m$
      Compute gradient estimation $g \leftarrow \frac{1}{m}\nabla_\theta \sum_i (L(f(x_i; \theta), y_i))$
      Compute velocity $\vartheta \leftarrow \alpha\vartheta - \epsilon g$
      Apply parameter update $\theta \leftarrow \theta + \vartheta$
  end while
```

2.1.2 Adaptive Learning Rate Optimizers

Learning rate is the most sensitive parameter that strongly affects performance of a model. It is also known that model cost highly relies on some parameters from model's parameter space. This brings up the idea of using separate learning rates per parameter based on their sensitivity, also adapting per parameter learning rates automatically. Earlier heuristic is increasing the learning rate if partial derivative sign of loss with respect to a model parameter remains the same, oppositely, decreasing the learning rate in case of gradient changes sign. Adaptive learning rate optimizers based on this heuristic are (mostly used examples) AdaGrad[16], Rmsprop[11] and Adam[12] optimizers. AdaGrad, an earlier example of adaptive learning rate optimizers, scales learning rate per parameter according to the square root of the sum of all historical squared values of the gradient. In this approach, parameters with larger partial derivatives have accelerated decrease compared to parameters with small partial derivatives. Even though this approach helps some models, keeping historical partial derivatives from beginning of training to end can introduce earlier and excessive decrease in the effective learning rate for some cases. Rmsprop modifies AdaGrad algorithm to address its excessive decrease on learning rate by weighted moving average of gradient history. Instead of AdaGrad, Rmsprop keeps moving average of gradients history with a new hyper-parameter that adjusts how historical gradients impact moving average besides current gradient. Aside from AdaGrad and Rmsprop, Adam optimizer combines Rmsprop and momentum optimizers benefits. Adam optimizer computes two historical moving average estimates which keep gradients and squared of gradients respectively. There is also correction of initial bias of moving average of gradients and square of gradients in Adam that is also an addition to Rmsprop algorithm.Although Adam is capable to work with different models with default values, some cases still requires global learning rate and other hyper-parameters tuning.

Algorithm 2 Adam Algorithm

```
Require: Learning rate $\epsilon$ (0.001 default)
Require: Exponential rate decay constants for moment estimates $p_1, p_2 \in (0, 1]$
  (0.9, 0.999 defaults)
Require: Small constant $\delta$ for stabilization ($10^{-8}$ default)
Require: Initial parameters $\theta$
  Initialize 1st and 2nd moments $s \leftarrow 0, r \leftarrow 0$
  Initialize time $t \leftarrow 0$
  while training stopping criterion not met do
      Sample a mini-batch of m examples from training set: $x_1, x_2 \dots x_m$ with
      corresponding labels $y_1, y_2 \dots y_m$
      Compute gradient estimation $g \leftarrow \frac{1}{m}\nabla_\theta \sum_i(L(f(x_i;\theta), y_i))$
      $t \leftarrow t + 1$
      Update biased 1st moment: $s \leftarrow p_1 s + (1 - p_1)g$
      update biased 2nd moment: $r \leftarrow p_2 r + (1 - p_2)g \odot g$
      Correct bias in 1st moment: $\hat{s} \leftarrow \frac{s}{1-p_1^t}$
      Correct bias in 2nd moment: $\hat{r} \leftarrow \frac{r}{1-p_2^t}$
      Compute update: $\Delta\theta \leftarrow -\epsilon\frac{\hat{s}}{\sqrt{\hat{r}}+\delta}$
      Apply parameter update $\theta \leftarrow \theta + \Delta\theta$
  end while
```

Rmsprop and Adam seem to be the most widely used adaptive learning rate based optimizers in deep learning community.

2.1.3 Revisiting SGD with Momentum

Common knowledge on deep learning training suggests to use adaptive learning rate based methods such as Adam and Rmsprop due to their robust performance over different learning rates. Regular user can obtain acceptable results without tuning learning rate. Even though adaptive methods are quite useful, there is still a remarkable difference between performance of model that uses the optimal learning rate and the default learning rate of adaptive methods. Recent researches show that SGD with momentum can obtain better test results over adaptive methods. Wilson et. al.[13] conduct a comprehensive study which shows that SGD and SGD with Momentum outperform on unseen dataset in designed tasks of over-parameterized models contrary to common belief on adaptive optimizers. Their findings favors SGD and SGD with momentum because adaptive methods find solutions having worse generalization than non-adaptive methods. Adaptive methods have faster progress during the earlier epochs of training while their final performance on test set is not promising. They also found out that tuning Adam classifier gains considerable improvement compared to default settings. Another research inspired from Wilson et. al. [13] findings, suggests a simple idea that using Adam on earlier epochs of deep neural network training then switching to SGD with Momentum to solve saturation of Adam on later epochs [17]. They showed using Adam and SGD with momentum instead of only using Adam results in better generalization. Several researches focus on correcting the generalization problem of Adam optimizer by adapting Adam algorithm. In [18] study, YOGI algorithm is proposed to solve this problem as a new adaptive optimizer. Their improvement applied to Adam optimizer is the controlled effective learning rate increase instead of Adam's rapid increase that leads better generalization. Similarly, Chen and Gu propose Partially adaptive momentum estimation method (PADAM) [19]. PADAM follows the idea of unifying Adam algorithm's fast convergence rate and good generalization of SGD with momentum for the final model that means combining best of both optimization algorithms. In short, PADAM uses a partial adaptivity parameter to control the adaptivity level. Particularly, If partial adaptivity parameters selected as close to zero, algorithm behaves like SGD with momentum, oppositely behaves more like Amsgrad that is one of variation of Adam algorithm. Since all these methods introduces new parameters and complexity, usage of these Adam style algorithms are still limited and not effective. Moreover, Zhang et al. [14] conduct several experiments that compares hand tuned SGD with momentum and adaptive optimization methods from this trend. According to their study, models trained with hand-tuned learning rate using SGD with momentum achieves faster convergence than Adam for many models. Indeed, many recent state-of-the-art image classification models on popular datasets, such as SVHN, CIFAR10 and ImageNet, report their results on SGD with momentum [20] [21] [2] [1].

2.2 Challenges on Neural Network Training

2.2.1 Achieving Good Generalization

Neural networks are powerful classifiers with the capability of learning complex patterns from training data through their hidden layers and large amount of neurons. These patterns usually include noise in training data that causes lower classifier performance on unseen data. Fitting model to training data too much that results in poorly generalized models is called overfitting problem. One of common approaches to reduce overfitting is monitoring training phase using validation dataset to stop training before overfitting starts.

- **Early Stopping and Cross-Validation:** A small part (typically %10-30) of training set is reserved as validation dataset which is used to detect when overfitting starts. When validation loss, i.e. error on unseen examples, starts to increase during training, training should be stopped to prevent overfitting. Even though early stopping solves generalization problem, it is not practical. Firstly, this method requires to sacrifice a part of training data to create a validation dataset. Secondly, it is hard to decide which examples should be in training dataset or validation dataset on real systems. These two datasets should be uniform in terms of hardness of examples, also; we don't want to lose informative examples by putting on validation dataset. To achieve more reliable validation, cross-validation technique is generally used. Cross-validation is based on the idea of assessing the performance on unseen data as a validation set, i.e. randomly chosen partition of training set. To reduce variability, multiple experiments are usually performed using different validation sets. Even if cross-validation allows reliable decision making with early stopping, those repetitive experiments can be costly in terms of time and resources.

- **Validation on Mini-batches:** One of the recent studies extends cross-validation idea that is to validate group of mini-batches with another mini-batch, that is called validation mini-batch, during the training procedure [22]. In their work, they formalize this procedure as dot product of calculated gradients of each training mini-batch and validation mini-batch. This calculation is used to determine the weight of the gradient update. If gradients of training mini-batch and validation set is correlated, then weight of this gradient update is large and positive. This can be considered as gradient update validation of a mini-batch update by another mini-batch to decide enlarging the update or discard it by calculating zero weight. Note that validation mini-batch is not a separate data chunk, contrary, validation mini-batch is directly participating to training with gradient updates. They show that their procedure results in better generalization of model, i.e. reducing overfitting. They explain this achievement as validation of gradients helps to avoid memorization by encouraging model parameter updates that only reduce errors on shared sample patterns.

2.2.2 Selecting Optimal Learning Rate

Selection of learning rate is remarkably important to accomplish the highest accuracy of a particular neural network architecture. Researchers have been explained the importance of using optimal learning rate and shared practical recommendations and techniques to decide correct learning rate for experiments of manually setting learning rate and decay. Specifically, many practical suggestions for manual tuning of learning rate are presented with the emphasis on careful selection of learning rate that is directly related with effective model capacity on [23] [24] [25]. Below we summarize why learning rate selection might be challenging for neural networks.

- Small learning rate can lead to horribly slow convergence while big learning rate can cause excessive fluctuations on loss curve which end up with saddle points.
- Finding optimal learning rate is not sufficient for many training tasks. Learning rate schedules or learning rate decay is usually required to reach best results

because training generally needs large steps on earlier epochs and smaller steps on later epochs.

- Different neural network architectures and datasets require different learning rates as optimal learning rate and schedule.

- Network architecture and dataset are not only factors that affect optimal learning rate, mini-batch size is directly related with optimal learning rate. Smaller mini-batch size usually requires smaller learning rate since gradients generated from smaller mini-batches are more noisy. In fact, different mini-batch size have different optimal learning rates even all other settings are the same for a particular neural network. Moreover, Smith et al. [26] shows that decaying learning rate and increasing mini-batch size during training can obtain the same error curves that shows strong dependency between two hyper-parameter.

- Researchers usually utilise their knowledge based on prior experiments for a particular dataset and/or neural networks. Although common benchmark datasets may provide extensive literature and prior experiments, newly introduced network models requires tuning carefully.

Considering these challenges to obtain optimal results on SGD training with neural networks, enormous number of experiments with selection of learning rate and learning rate annealing schedules require considerable amount of time. Deep learning researches focus on automated tuning of learning rate instead of these manual efforts. Adaptive learning rate methods are designed to tune learning rate for each individual parameter on the fly such as Adam, Rmsprop and AdaGrad etc. which are briefly mentioned in Section 2.1.2. These methods seems to take the burden of manual learning rate tuning process because they generally work well with their default parameters. However, recent researches point out that SGD with momentum can attain better performance with ideal learning rate and decay schedules. This finding brings up complex schedules of SGD with momentum. The interest on SGD with momentum motivates us to automatize learning rate tuning process. Our proposed method will be explained step by step on Chapter 3. Other learning rate tuning methods are discussed in Section 2.3

2.3 Learning Rate Search

Many methods have been proposed for automated learning rate selection [27, 28, 29, 30, 14, 31, 32]. The ultimate goal is to obtain lowest generalization error in a minimum time and memory budget. Since importance of optimal learning rate and learning rate schedule has been known, also discussed in Section 2.2.2, for a long time, many researches have been focused on automatized approaches for this purpose. This chapter covers traditional hyper-parameter tuning approaches, cyclical learning rate schedules and gradient based learning rate tuning methodologies as a background of learning rate tuning.

2.3.1 Automated Hyper-parameter Tuning

Automated hyper-parameter tuning can be considered as ancestor of automated learning rate tuning. There are two main approach in hyper-parameter tuning that are sequential and parallel tuning. Sequential methods are similar to manual learning rate tuning that obtain a model with one of hyper-parameter set, performance of this model directs the subsequent selection of hyper-parameters. The disadvantage of sequential tuning is time consumption like manual tuning. However, parallel tuning creates multiple models of different hyper-parameter settings in parallel that seems to solve time cost problem while this introduces a computational resource problem which is also costly. Known examples of parallel tuning are grid search and random search. Grid search is a hyper-parameter optimization method for machine learning problems. Although we focus on learning-rate optimization, our problem can be extended to hyper-parameter optimization problem which has been studied over the decades. Number of hidden layers, learning rate, regularisation strength and mini-batch size etc. are hyper-parameters that are needed to be tuned for neural networks. Grid search is a naive exhaustive search method which requires a set of values defined by experimenter. In grid search, every possible combination of hyper-parameters forms an experiment trial which grows exponentially with the number of hyper-parameters. If the number of parameters and search space is small, grid search would perform in a reasonable time.

While Grid search and manual search are widely used on hyper-parameter optimization of machine learning problems -also combined- random search is able to find models that have as good or better performance over the same search domain. Bergstra and Bengio[27] propose the random search which is more efficient in high dimensional spaces of hyper-parameters. Instead of combining all possible values of hyper-parameters for trials, random search forms experiments randomly from their search space. Their motivation is that final performance of neural networks mostly rely on some hyper-parameters more than others that limits grid search because unimportant hyper-parameter limits by repeating trials and limiting the important hyper-parameter search space. However, they also report that 32 dimensional search problem of deep belief network (DBN) optimization is not as good as sequential combination of manual and grid search of an expert. However, both random search and grid search are brute-force approaches, they scale exponentially with the number of hyper-parameters. Even if the researcher performs experiments on distributed systems in parallel, increase in hyper-parameter dimension can easily exceed resource limits due to excessive number of parallel runs.

2.3.2 Cyclical Learning Rates

Cyclical learning rates (CLR) method [28] aims to resolve learning rate and learning rate schedule tuning problem. The method proposes to train neural network by cyclically varying learning rates within the predefined boundaries instead of decreasing the learning rate. Boundaries of cyclical learning rate can be found by linearly increasing the learning rate of the network for a few epochs which is called "LR range test". The optimal learning rate is inside these boundaries. Then, learning rates when model accuracy starts to increase and to decrease are taken to use as boundaries of cycle. Learning rate varies between minimum to maximum boundary and maximum to minimum boundary in a defined step size during training. These method also requires the selection of step size i.e number of iterations to take a half learning rate cycle from minimum to maximum. Cycle topology can be triangular or exponential. CLR experiments of different neural network architectures show that CLR achieves better accuracy than fixed learning rate methods. Compared with traditional way of decreasing learning rate over steps, why CLR methods work is explained in two factors:

- Learning rate boundaries include optimal value for the task which helps CLR to use optimal and closer values to optimal learning rate.
- Minimizing loss around saddle points is difficult because they have small gradients that slows down the learning, larger learning rates can help to avoid saddle points.

In the follow-up research of CLR, Smith et al. [29] propose super convergence phenomenon that can be achieved using very large learning rates on cyclical learning rate method. They present that using large learning rates helps to regularize the network that result in better model performance. In learning rate range test, increasing learning rate causes an increase on training loss while test loss is surprisingly decreased at the same time for Resnet-56 architecture. However, proposed rapid convergence only showed in a single task for this method.

Another learning rate decaying method similar to cyclical learning rates is SGD with warm restarts (SGDR) [30]. In this work, Loshchilov et al. proposes to initialize learning rate to some value that is scheduled to decay with an aggressive cosine annealing schedule. Warm restarts refers to restarting only learning rate while other model parameters remain the same with the latest step. SGDR improves many state-of-the-art models error rates on popular datasets. These methods are important since they show increasing learning rate from time to time to reasonably higher values can be beneficial for final network performance by finding flat local minima. Disadvantage of these automated schedules is finding learning rate boundaries as minimum and maximum learning rate still requires additional efforts to achieve optimal performance.

2.3.3 Gradient Based Tuning Methods

Another automated learning rate adjustment approach is using gradients to decide learning rate. Several researches have focused on using gradient information for automated learning rate tuning. In Schaul et al.'s work [31], they define a formula to obtain the optimal learning rates for SGD using estimates of the variance of the gradients. This method can find either single global learning rate or learning rates of each parameter and parameter groups. Their method defines a formula that is adapting "bbprop" which computes the positive estimates of diagonal Hessian terms for single sample using back propagation [33]. Moving averages of gradients and diagonal Hessian are used to estimate learning rate in their algorithm. Thus, they obtained an algorithm that automatically decreased learning rate to zero when loss function is approaching to its optimal value without any manual learning rate search. Zhang. et al. [14] conduct experiments to show that carefully hand-tuned learning rate can be competitive with adaptive learning rate optimizers. Their study is not only analyzing this phenomena but also, they come up with an automated learning rate and momentum tuning approach called Yellowfin. They consider learning rate tuning problem together with momentum tuning. Similar to Schaul et al.'s work [31], they use noisy quadratic model. In their tuner, hyper-parameters are tuned in every training step using curvature range and gradient variance estimates. Another work proposes hypergradient descent method to tune learning rate [32]. Apart from historical background of hypergradient that previously considered in optimization literature, they define hypergradient descent as applying gradient descent to learning rate in each training step. This means calculating the partial derivative of the objective function at the previous time step with respect to the learning rate. Hypergradient descent applied on SGD, SGD with Nesterov momentum and Adam is showing that need for manual tuning is reduced.

CHAPTER 3

MICRO CROSS VALIDATION ALGORITHMS

In this chapter, micro cross validation algorithms that we propose are described. We propose two main types of MCV methods. The first type is training step verification using MCV that accepts or rejects training steps. This method is running with given learning rate and its schedule (Section 3.1). The second type is automated search of learning rate. This type applies adaptive search of learning rates using different loss functions which takes into account the validation batch loss and various combinations of validation and training batch losses (Section 3.2). This chapter includes motivation and algorithmic details of micro cross validation algorithms, more specifically training step verification using micro cross validation and automated learning rate search using micro cross validation.

3.1 Training Step Verification Using Micro Cross Validation

The motivation behind Micro Cross-Validation (MCV) is to reduce user's effort on training because to obtain a model which reaches its effective capacity so that performs well on unseen data. Learning noisy patterns from training data and fitting training data too much is known as overfitting problem which is explained in Section 2.2.1.

Training step verification using micro cross-validation intends to remove gradient updates which are not verified by another random mini-batch, i.e. observing decrease on loss of second random mini-batch with the update of first random mini-batch gradient. MCV approach can be considered as batch-level version of classical cross-validation that requires splitting a validation dataset from training dataset randomly by monitoring model generalization on this validation set. Instead of creating separate validation dataset for multiple times, we propose to apply validation on a random mini-batch in every training step that can also be a part of training set in latter epochs.

MCV is different from the standard neural network training process in the following aspects:

- In each training step, single mini-batch is used for gradient update step while another random mini-batch is used for validation of this update. In each epoch, training data is shuffled to make sure that different data points can be chosen as training or validation mini-batches in different epochs. So, validation mini-batches are not obtained from separate dataset, contrary, validation mini-batches are directly used on gradient calculation and training updates in subsequent epochs.
- Since splitting a validation dataset from training dataset can be biased in terms of hardness of examples and noise of examples, validation dataset is created multiple times using different examples from training dataset. So, training and evaluation of model performance on validation dataset is repeated multiple times that causes excessive time cost. MCV solves this problem since single run of MCV uses whole training data and performs validation on different random mini-batches of training dataset in every step.
- Apart from classical validation and monitoring of model performance, MCV directly changes the updates of training by validating each update to determine whether they are found as useful or not. Specifically, after each gradient update, the loss of validation mini-batch is calculated. If loss value of validation mini-batch is not decreased from previous value that is calculated before the gradient update step, this gradient update is discarded by reloading previous model parameters and state. So, all gradient updates are not applied to model unlike regular training process.

Difference between macro cross-validation and micro cross-validation is presented in Figure 3.1. Algorithm 3 shows how MCV is applied to the SGD with momentum method.

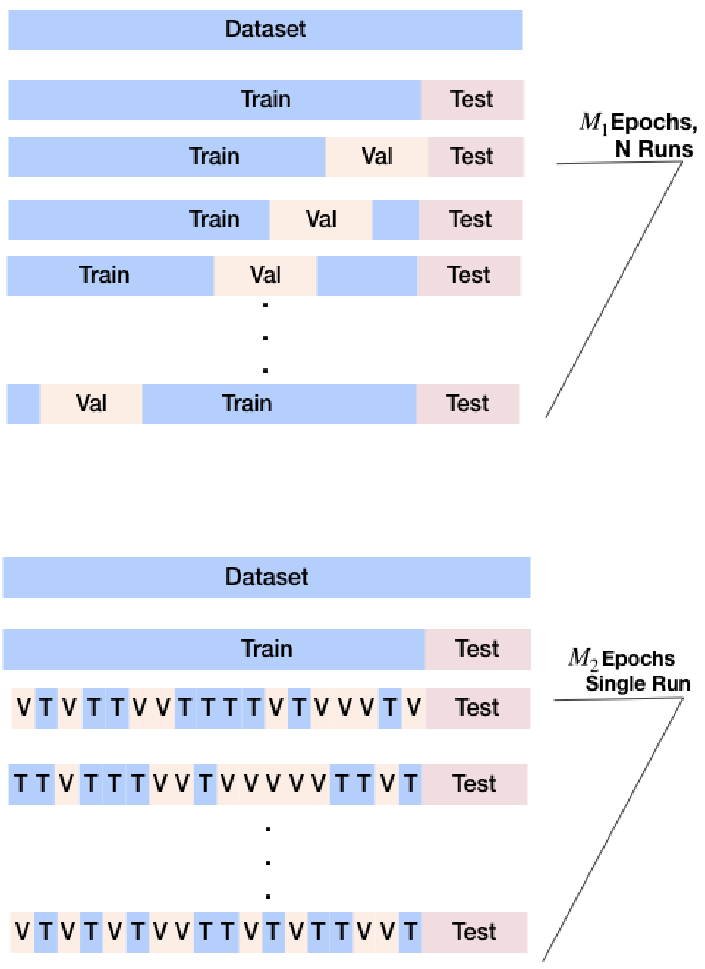

Figure 3.1: Cross-validation uses different splits of validation dataset to monitor training and tune parameters of model with multiple runs. In each run, different part of train dataset is used as validation dataset that cannot be used on training updates. Since splitting dataset creates randomness, average of multiple runs is required for stable results. Micro cross-validation randomly selects mini-batches for validation in each epoch. In each training step, randomly selected validation mini-batch used for validating training update of a train mini-batch. After an epoch is completed, dataset is shuffled so that different train and validation mini-batches are constructed. That means a data sample can be in both training set and validation set in different epochs. Unlike cross-validation, single run of MCV is enough because all of dataset is used during training.

Algorithm 3 Training Step Verification using Micro Cross-Validation for SGD with Momentum

```
while training stopping criterion not met do
    Sample a mini-batch of m examples from training set: $x_{11}, x_{12} \ldots x_{1m}$ with corresponding labels $y_{11}, y_{12} \ldots y_{1m}$ called as $batch_1$
    Sample a mini-batch of m examples from training set: $x_{21}, x_{22} \ldots x_{2m}$ with corresponding labels $y_{21}, y_{22} \ldots y_{2m}$ called $batch_2$
    Calculate loss of $batch_2$ : $current\_loss\_batch_2 \leftarrow \frac{1}{m} \sum_{2i}(L(f(x_{2i}), y_{2i}))$
    Back up parameter values of model and velocity buffer:
    $prev\_parameters \leftarrow \forall\theta \in model$
    $prev\_\vartheta \leftarrow \vartheta$
    Compute gradient estimation of $batch_1$ : $g \leftarrow \frac{1}{m}\nabla_\theta \sum_{1i}(L(f(x_{1i};\theta), y_{1i}))$
    Compute velocity $\vartheta \leftarrow \alpha\vartheta - \epsilon g$
    Apply parameter update $\theta \leftarrow \theta + \vartheta$
    Calculate new loss of $batch_2$ : $new\_loss\_batch_2 \leftarrow \frac{1}{m} \sum_{2i}(L(f(x_{2i}), y_{2i}))$
    if $new\_loss\_batch_2 > current\_loss\_batch_2$ then
        Discard latest gradient update by restoring previous model and velocity:
        $\vartheta \leftarrow prev\_\vartheta$
        $model.parameters \leftarrow prev\_parameters$
    end if
end while
```

3.2 Automated Learning Rate Search Using Micro Cross-Validation

Automated learning rate search using MCV method aims to improve training step verification by applying learning rate selection based on random validation mini-batch in each training step automatically. Instead of simply accepting or discarding training steps based on validation mini-batch, this method tunes learning rate using MCV. Optimal learning rate and decay schedule selections are crucial for neural network training performance. Several automated learning rate tuning methods are mentioned in Section 2.3. Our method tries to make optimal learning rate selections for each training step that is defined as learning rates providing smallest loss of validation batch after each mini-batch update. To extend this idea, we proposed a history based voting mechanism which keeps exponential moving average of each training step's learning rate selection based on second validation mini-batch. Exponential moving average (EMA) is a moving average the weight of each item decreases exponentially by time (i.e. iterations) and never reaches to zero. The EMA of series of items can be calculated as:

$$S_t = \begin{cases} Y_1 & t = 1 \\ \alpha . Y_t + (1-\alpha) . S_{t-1} & t > 1 \end{cases}$$

The α coefficient represents the degree of weighting weather it is fast or slow.It should be between $(0, 1]$. If α is closer to 1, older values have faster decreasing effect on the moving average.

Following concepts are used to construct our naive learning rate search algorithm (i.e. MCV-lr-voting):

- **Learning rate list** : Learning rates that will be used in model training should be defined at the beginning. Instead of common convention, that is setting single optimal learning rate and its decay schedule at the beginning of training, array of learning rates can be initialised at the beginning. Our method tries to select best learning rate from this list based on validation mini-batch's loss.

- **Best learning rate** : In each training step, one mini-batch for gradient update and one another mini-batch for validation is sampled from training set. Lets say they are $batch_1$ and $batch_2$ respectively. After gradient calculated using $batch_1$, gradient update step with each learning rate in learning rate list is applied on model then $batch_2$ loss is calculated for each training step. Learning rate that most decreases loss of $batch_2$ is selected as best learning rate of this training step. If none of learning rates manages to decrease $batch_2$ loss, best learning rate selected as zero which means a discard on this step. This learning rate selection and $batch_2$ loss comparison is completely fair because after each update, previous model parameters and velocity array are reloaded.

- **Voting and learning rate history**: In each training step, best learning rate is found using $batch_1$ and $batch_2$. Best learning rate is not used directly for final gradient update, instead, they are used to calculate exponential moving decay of learning rates. Each best learning rate is a vote on learning rate history. This vote updates the learning rate history using EMA formula:
  $history_t = \alpha.vote_t + (1 - \alpha).history_{(t-1)}$

- **Top learning rate**: Best learning rate is used as a vote on learning rate history. After update of learning rate history with selected best learning rate, learning rate that have highest value on history is called top learning rate. Gradient update with $batch_1$ is applied with top learning rate in each step. To sum up, actual learning rate used in the training step is found by all learning rate selections from the beginning of training to current step. Due to EMA, effect of very earlier training steps are decreased over time.

Algorithm 4 Learning Rate Search Using Votes of Micro Cross-Validation

```
Require: Learning rate $\epsilon$, momentum parameter $\alpha$
Require: Initial parameter $\theta$, initial velocity $\vartheta$
Require: learning rate selection list: $lr\_list$    ▷ typically $\{0.1, 0.01, 0.001, 0.0001\}$
  Initialize: $lr\_history \leftarrow \{0.25, 0.25, 0.25, 0.25\}$    ▷ uniformly in size of $lr\_list$
  Initialize: $best\_loss \leftarrow \inf$
  while training stopping criterion not met do
      Sample a mini-batch of m examples from training set: $x_{11}, x_{12} \ldots x_{1m}$ with
      corresponding labels $y_{11}, y_{12} \ldots y_{1m}$ called as $batch_1$
      Sample a mini-batch of m examples from training set: $x_{21}, x_{22} \ldots x_{2m}$ with
      corresponding labels $y_{21}, y_{22} \ldots y_{2m}$ called $batch_2$
      $best\_lr \leftarrow FindBestLR(batch_1, batch_2, lr\_list)$
      $lr\_history \leftarrow UpdateHistory(best\_lr)$
      $\epsilon \leftarrow TopLR(lr\_list, lr\_history)$
      Compute velocity $\vartheta \leftarrow \alpha\vartheta - \epsilon g$
      Apply parameter update $\theta \leftarrow \theta + \vartheta$
  end while
```

Algorithm 5 Find Best Learning Rate for Training Step

```
Input: $batch_1 : x_{11}, x_{12} \dots x_{1m}$ and $y_{11}, y_{12} \dots y_{1m}$
Input: $batch_2 : x_{21}, x_{22} \dots x_{2m}$ and $y_{21}, y_{22} \dots y_{2m}$
Input: learning rate selection list: $lr\_list$        ▷ typically $\{0.1, 0.01, 0.001, 0.0001\}$
  Calculate loss of $batch_2 : current\_loss \leftarrow \frac{1}{m} \sum_{2i}(L(f(x_{2i}), y_{2i}))$
  Compute gradient estimation of $batch_1 : g \leftarrow \frac{1}{m} \nabla_\theta \sum_{1i}(L(f(x_{1i}; \theta), y_{1i}))$
  Back-up model parameters and momentum velocity:
  $prev\_parameters \leftarrow \forall \theta \in model$
  $prev\_\vartheta \leftarrow \vartheta$
  for all $l$ such that $l \in lr\_list$ do
      $\epsilon \leftarrow l$
      Compute velocity $\vartheta \leftarrow \alpha\vartheta - \epsilon g$
      Parameter updates $\theta \leftarrow \theta + \vartheta$ , $\forall \theta \in model$
      Calculate new loss of $batch_2 : new\_loss \leftarrow \frac{1}{m} \sum_{2i}(L(f(x_{2i}), y_{2i}))$
      if $(new\_loss < current\_loss)$ & $(new\_loss < best\_loss)$ then
          $best\_lr \leftarrow l$
          $best\_loss \leftarrow new\_loss$
      end if
      Discard latest gradient update by restoring previous model and velocity:
      $\vartheta \leftarrow prev\_\vartheta$
      $model.parameters \leftarrow prev\_parameters$
  end for
  return $best\_lr$
```

Algorithm 6 Update Learning Rate History

```
Input: learning rate selection list: $lr\_list$        ▷ typically $\{0.1, 0.01, 0.001, 0.0001\}$
Input: learning rate history: $lr\_history$, $best\_lr \in lr\_list$
  $votes \leftarrow \{lr_1 \leftarrow 0, lr_2 \leftarrow 0 \dots lr_n \leftarrow 0\}$
  $votes\,\{best\_lr\} \leftarrow 1$
  $lr\_history \leftarrow \alpha.votes + (1 - \alpha).lr\_history$
  return $lr\_history$
```

Final proposed learning rate search algorithm is adaptive learning rate search using loss logs of MCV. This algorithm is different in terms of learning rate varieties respect to learning rate search using votes of MCV algorithm. Unlike naive learning rate search, this algorithm selects best learning rate based on average of validation losses. Then, new learning rate selection list is constructed by following formula:
$lr\_list = \left[5lr, 2lr, \frac{4}{3}lr, lr, \frac{3}{4}lr, \frac{1}{2}lr, \frac{1}{5}lr\right]$
By this formula, all possible values between minimum and maximum learning rate can be used instead of predefined 4-5 learning rates. Adaptive learning rate search using loss logs of MCV algorithm calculates average of validation loss values obtained from micro cross-validation to select learning rate for each epoch. For this algorithm using validation loss is a general idea but we also test the effect of train loss after each step of candidate learning rate. Different combinations of validation loss of $batch\_2$ and training loss of $batch\_1$ are considered. The formula for logging losses is:
$(1 - \alpha) * val\_loss + \alpha * train\_losses, \alpha \in [0, 0.1, 0.3, 0.5]$

For this algorithm (Algorithm 8), selection of learning rate strategy is defined in two different approaches:

- Learning rate having minimum validation loss is selected.
- Learning rates whose average validation loss is lower than the average loss of validation batch before training update, i.e. $avg\_current\_loss$, are collected. Then, the largest valued learning rate of them is selected. If none of learning rate can decrease validation loss in average, learning rate having minimum validation loss is selected like first approach.

Stopping criteria of adaptive learning rate search using loss logs of MCV is defined as following steps:

- If average validation loss of epoch is not decreased in pre-defined epoch limit, i.e. patience, training is stopped.
- If average validation loss of epoch is none because of exploding gradients problem comes from bigger learning rates, training is stopped and epoch of last recorded minimum average validation loss is selected as stopped epoch.

- If first and second criterias are not executed, training is done with pre-defined epoch number.

Algorithm 7 Collect Losses for Learning Rate Selection

```
Input: $batch_1 : x_{11}, x_{12} \ldots x_{1m}$ and $y_{11}, y_{12} \ldots y_{1m}$
Input: $batch_2 : x_{21}, x_{22} \ldots x_{2m}$ and $y_{21}, y_{22} \ldots y_{2m}$
Input: learning rate selection list: $lr\_list$        ▷ typically $\{0.1, 0.01, 0.001, 0.0001\}$
  Initialize: $losses \leftarrow \{\}$
  Calculate loss of $batch_2 : current\_loss \leftarrow \frac{1}{m} \sum_{2i}(L(f(x_{2i}), y_{2i}))$
  Compute gradient estimation of $batch_1 : g \leftarrow \frac{1}{m} \nabla_\theta \sum_{1i}(L(f(x_{1i}; \theta), y_{1i}))$
  Back-up model parameters and momentum velocity:
  $prev\_parameters \leftarrow \forall\theta \in model$
  $prev\_\vartheta \leftarrow \vartheta$
  for all $l$ such that $l \in lr\_list$ do
      $\epsilon \leftarrow l$
      Compute velocity $\vartheta \leftarrow \alpha\vartheta - \epsilon g$
      Parameter updates $\theta \leftarrow \theta + \vartheta$ , $\forall\theta \in model$
      Calculate new loss of $batch_2 : new\_loss \leftarrow \frac{1}{m} \sum_{2i}(L(f(x_{2i}), y_{2i}))$
      $losses \leftarrow losses + \{new\_loss\}$
      Discard latest gradient update by restoring previous model and velocity:
      $\vartheta \leftarrow prev\_\vartheta$
      $model.parameters \leftarrow prev\_parameters$
  end for
  return $losses, current\_loss$
```

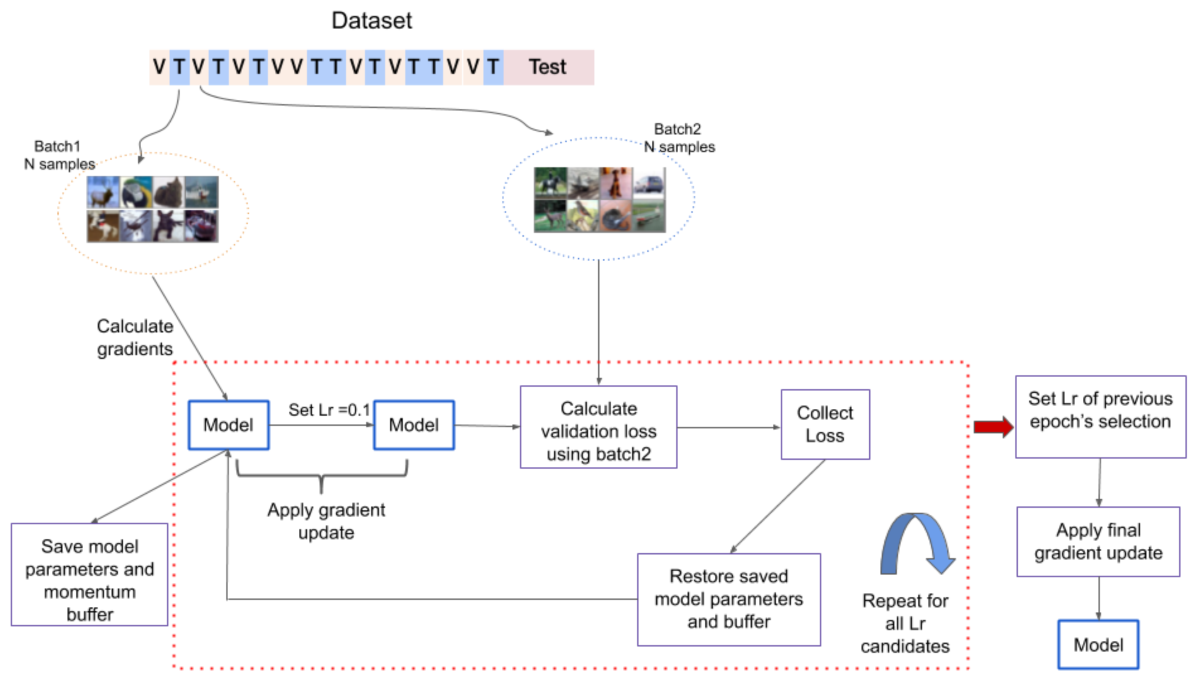

Figure 3.2: In each training step, training mini-batch, i.e. $batch\_1$ and validation mini-batch, i.e. $batch\_2$, are sampled from training dataset. Gradients of model parameters are calculated respect to $batch\_1$ over objective function. Gradients will be used for training update step of each candidate learning rates. All model parameters are backed up for further model restore at the end of each learning rate trial step. A learning rate is set to the model's optimizer to perform training update. Then, validation loss of $batch\_2$ is calculated on updated model which is collected for latter learning rate decision. Some of loss logging rules also include training loss of $batch\_1$ with combination with validation loss of $batch\_2$. Then, model parameters and optimizer momentum buffer are restored with the backed up model. When all validation losses of candidate learning rates are collected for learning rate decision at the end of epoch, actual training step is applied with the selected learning rate of previous epoch. At the end of each epoch, learning rate is selected based on average losses of learning rates.

Algorithm 8 Adaptive Learning Rate Search Using Loss Logs of Micro Cross-Validation

```
Require: Learning rate $\epsilon$, momentum parameter $\alpha$
Require: Initial parameter $\theta$, initial velocity $\vartheta$
Require: learning rate selection list: $lr\_list$    ▷ typically $\{0.1, 0.01, 0.001, 0.0001\}$
  Initialize: $losses \leftarrow \{\}$ , $lr \leftarrow min(lr\_list)$
  for each $epoch \in total\_epochs$ do
    if $epoch > 1$ then
      $lr \leftarrow SelectLR(avg\_losses, current\_loss, lr\_list)$
      $lr\_list \leftarrow \left[5lr, 2lr, \frac{4}{3}lr, lr, \frac{3}{4}lr, \frac{1}{2}lr, \frac{1}{5}lr\right]$
    end if
    $avg\_losses \leftarrow \{0, 0, 0, 0, 0, 0, 0\}$
    $avg\_current\_loss \leftarrow 0$
    while sample mini-batches $batch\_1, batch\_2 \in training\_set$ do
      $losses \leftarrow CollectLosses(batch_1, batch_2, lr\_list)$
      $avg\_losses \leftarrow avg\_losses + losses$
      $avg\_current\_loss \leftarrow avg\_current\_loss + current\_loss$
      $\epsilon \leftarrow lr$
      Compute velocity $\vartheta \leftarrow \alpha\vartheta - \epsilon g$
      Apply parameter update $\theta \leftarrow \theta + \vartheta$
    end while
    $avg\_losses \leftarrow avg\_losses/batch\_num$
    $avg\_current\_loss \leftarrow avg\_current\_loss/batch\_num$
  end for
```

CHAPTER 4

EXPERIMENTS

4.1 Introduction

In this chapter, we present our experimental results for the micro (i.e. batch-level) cross-validation (MCV) methods described in Chapter 3. We designed our experiments to answer the central questions that we pose in the Introduction section of this thesis:

- **[Q1]** In a setting where learning rate and its decay schedule are given, does micro cross-validation help improve generalization? Does it yield a better test set accuracy?
- **[Q2]** In what ways can we use micro cross-validation to automate the search for optimal learning rate and learning rate schedule?
- **[Q3]** Do MCV methods that automatically select learning rates improve test set accuracy over baseline methods?

Exploring these questions on a single dataset using a single network architecture would give us a limited picture. In order to increase the generality of our answers, in our experiments, we used three image datasets, two of them being widely used small-scale benchmark datasets (CIFAR-10 [34] and SVHN [35]) and one of them being a larger scale dataset for age prediction from face images (Adience [36]). We evaluated three different convolutional neural network (CNN) architectures: a small, custom CNN, a ResNet[20] and a VGG[37] network.

To explore answers to the above questions, we proposed various MCV methods. For

Q1, as done in the standard gradient descent optimization, we simply calculate the gradient but before doing the update, we test this gradient direction with the current learning rate on a random mini validation batch. If the loss is decreased on this validation batch, the gradient update is accepted, and otherwise it is rejected. For Q2, we proposed two different adaptive learning rate search methods. The first one computes a moving average of the votes for the best batch-level learning rates, and instantaneously uses the highest-voted learning rate for actual gradient updates. In the second method, a constant LR is used during an epoch and while doing so, average validation loss is computed per LR in the search array. At the beginning of a new epoch, we choose the best. For this second method, we proposed two different loss functions. These loss functions take into account the validation batch loss and various combinations of validation batch loss and training batch loss. To answer Q3, we used SGD with momentum learning method within a naive (grid) search cross-validation, early-stopping framework as our baseline method.

We describe the datasets in Section 4.2, the network architectures in Section 4.3 and performance metrics in Section 4.4. The rest of the sections present and discuss the results of our experiments.

4.2 Datasets

4.2.1 CIFAR-10

The CIFAR-10[34] dataset contains 60000 32x32 color images from 10 classes which are airplane, automobile, bird, cat, deer, dog, frog, horse, ship and truck. 50000 images used as training examples and remaining 10000 images used as test examples.

4.2.2 SVHN

The Street View House Numbers (SVHN) dataset [35] contains colored digits from Google Street View images which are obtained from real world street house numbers. We used the cropped and centered version of SVHN that contains 32x32 73257 training examples and 26032 test examples.

4.2.3 Adience

The Adience[36] dataset includes face images for the tasks of age and gender prediction. The dataset consists of 26,580 images from 2,284 subjects which are labeled to following 8 age interval classes ((0-2, 4-6, 8-13, 15-20, 25-32, 38-43, 48-53, 60-)). Cropped and aligned images are also provided which we used in our work. Provided train, test, validation sets include 11823, 4316 and 1284 images respectively. We performed only age classification task for Adience dataset.

4.3 Network Architectures

Experimented architectures of CIFAR-10 and SVHN, which are Resnet-18 and VGG-11, are demonstrated and explained in Figure 4.1. Also, we conduct experiments for small custom CNN architecture that have only convolutional and fully connected layers without any regularization such as batch normalization and dropout. Small CNN architecture for Cifar-10 includes six 3x3 convolutional layers which have 48, 48, 96, 96, 192, 192 output channels respectively. Convolutions are followed by three fully connected layers which are 512, 265, 10 output size respectively. Since digit classification is an easier task, we used two different small architectures for CIFAR10 and SVHN. Small CNN architecture of SVHN experiments has four 3x3 convolutional layers which have 32, 32, 64, 64 output channels respectively. Convolutions are followed by 2 fully connected layers, 512 and 10 size. Unlike CIFAR-10 and SVHN, Adience dataset is more challenging (it has high-resolution human face images). So, a large capacity neural network can be beneficial for this dataset's age classification task. So, we choose to use Resnet-50 architecture for Adience.

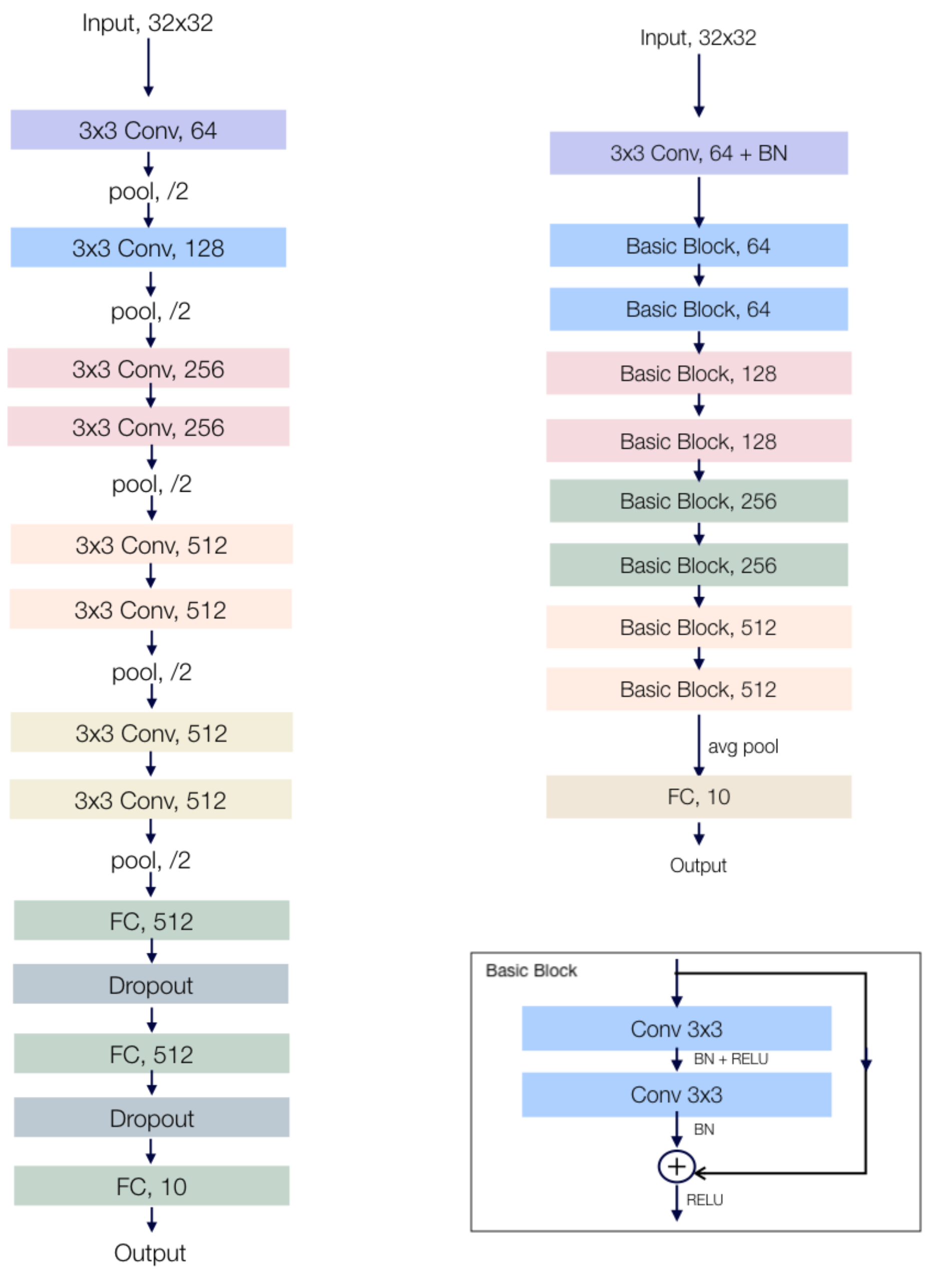

Figure 4.1: VGG-11 and Resnet-18 architectures are shown respectively. VGG-11 architecture consists of 3x3 convolutional layers followed by 3 FC layers with dropout layers. Resnet-18 architecture includes 3x3 convolutional layer followed by 8 basic blocks and single FC layer for output. Basic block can be shown bottom of Resnet-18 architecture that includes two 3x3 convolutional layers with identity shortcut connection which is added to stacked layers output. Note that only Resnet-18 architecture has batch normalization on convolutional layers.

4.4 Performance Metrics

Performance of baseline methods and automated adaptive learning rate search methods are also compared in terms of time cost. Two different performance metrics are used:

- **Running time of experiments:** Running (wall clock) time of baseline experiments and automated adaptive learning rate search experiments are reported for comparison. For the baseline, running time of all experiments for each learning rate and its decay settings are summed up to obtain total time passed while the best learning rate and its decay setting is found. For adaptive learning rate search methods, time cost of single execution is reported using average epoch counts. Multiple experiments of the same learning rate setting are ignored for baseline and our methods.
- **Theoretical complexity:** Complex operations of training, i.e. high time cost, are forward, backward and step operations. Forward operation refers to calculation of output layer's values through passing all neurons of DNN with input. A single forward is required for calculation of loss between output of network and real classes. Backward operation is performing backpropagation from networks last layer to the first layer by applying the chain rule to calculate gradients. After the backward operation, weights (i.e. parameters) of network are updated with step operation. Step operation calculates final parameter update inside optimizer with gradients (e.g. momentum buffer calculation for SGD with momentum). Table 4.1 shows the number of those operations per training step and explains those numbers with algorithmic details. An experiment's theoretical complexity is calculated with the following formula:
  $cost = avg\_epochs * steps\_per\_epoch * [(\#forward + \#backward), \#step]$

Table 4.1: Theoretical complexity of a training step for classical training and adaptive learning rate search methods are reported for comparison. These numbers are given for 7 learning rates in learning rate search list for querying. Classic training (baseline) requires single forward and backward followed by single step update on model weights. Even if classical training seems effective in terms of theoretical complexity, multiple runs for different learning rates result in poor performance at total. Since Method1 and Method3 collect validation losses of candidate learning rates, 7 forwards are applied in each step before the actual training step's forward operation. Also, current validation loss calculation (validation loss value before new step with candidate learning rate) adds one more forward operation that is 9 total forward operations per training step. Method 2 uses both training batch's loss and validation batch's loss of each candidate learning rate step operation. So, this adds 14 forward operations to the two forward operations (average current validation loss and actual step) that is 16 in total. Since gradients are re-used for each candidate step, backward count is 1 for all custom methods. Step counts of learning rate search methods are equal to the number of the candidate learning rate for search phase (which is 7 in our adaptive algorithms) and one more actual applied step that is total 8.

Method	**#Forward**	**#Backward**	**#Step Update**
Baseline	1	1	1
MCV-lr-voting	|LR|+2	1	|LR|+1
Method 1	|LR|+2	1	|LR|+1
Method 2	2|LR|+2	1	|LR|+1
Method 3	|LR|+2	1	|LR|+1

4.5 CIFAR-10 Experiments

We experimented with several different MCV methods on CIFAR-10 dataset using small CNN, Resnet-18 and VGG-11 architectures. Architectures are explained in Section 4.3 in detail. To compare our proposed methods with classical training, first, we obtain baseline results, we used early stopping for hyper-parameter tuning. Specifically, learning rates which are $\{0.1, 0.01, 0.001, 0.0001\}$ without any decay or complex schedule are evaluated using early stopping algorithm of patience of 20 epochs. Each learning rate is trained 5 times to obtain average results because each time we split training dataset into stratified random $\%80$ for training and $\%20$ for validation dataset that introduces randomness on results. Epoch count is decided from the first model run with early stopping. Second model using the same setting and $\%100$ of training dataset is trained with the epoch count found on first model run with early stopping. Second model is evaluated to obtain final test error value and accuracy. This dataset split introduce randomness between runs that requires multiple runs to obtain final learning rate and decay setting selection. The best resulting learning rate is selected as the base learning rate for further experiments with several learning rate decay selections. Learning rate decay is an exponential decay formulated as $initial\_lr * \frac{1}{1.+lr\_decay*step}$. Baseline runs are conducted separately for SGD with momentum and Adam optimizers. Since Adam uses implicit decaying, learning rate decay is not applied to experiments explicitly. Baseline results of all architectures are demonstrated in Table 4.2, 4.3 and 4.4.

Table 4.2: **CIFAR-10 baseline results on small CNN architecture.** Experiments with average epoch count, test accuracy and test loss that are conducted to obtain learning rate and learning rate decay setting of best accuracy result. Each setting of learning rate and learning rate decay is repeated 5 times due to randomness of %80-%20 training set, validation set splits for early stopping. The best resulting learning rate and learning rate decay will be used in further experiments of the same CNN architecture. Since none of regularization methods are not applied to network, overfitting starts quickly according to early stopping with 20 epoch patience factor. Reported epoch numbers in the table shows epochs that gives smallest validation error with a 20 more epoch training tries, i.e. 20 epoch patience of early stopping.

Optimizer	**LR**	**LR Decay**	**Epochs**	**Test Loss**	**Test Acc**
Momentum SGD	10^{-1}	–	3	1.77	33.5
Momentum SGD	10^{-2}	–	6	**0.72**	**76.37**
Momentum SGD	10^{-3}	–	16.6	0.86	72.23
Momentum SGD	10^{-4}	–	80	0.97	67.21
Momentum SGD	10^{-2}	10^{-3}	7.4	0.77	74.63
Momentum SGD	10^{-2}	5.10^{-4}	6.8	0.75	75.46
Momentum SGD	10^{-2}	10^{-4}	6.4	**0.72**	**76.6**
Adam	10^{-1}	–	13.8	2.31	10
Adam	10^{-2}	–	14.4	2.20	14.33
Adam	10^{-3}	–	4.8	**0.68**	**77.3**
Adam	10^{-4}	–	12	0.79	74.25

Table 4.3: **CIFAR-10 average baseline results on Resnet-18.** Experiments with average epoch count, test accuracy and test loss that are conducted to obtain learning rate and learning rate decay setting of best accuracy result. Each setting of learning rate and learning rate decay is repeated 5 times due to randomness of $\%80$-$\%20$ training set, validation set splits for early stopping. The best resulting learning rate and learning rate decay will be used in further experiments of the same Resnet-18 architecture. Since none of regularization methods are not applied to network, overfitting starts quickly according to early stopping with 20 epoch patience factor. Reported epoch numbers in the table shows epochs that gives smallest validation error with a 20 more epoch training tries i.e. 20 epoch patience of early stopping.

Optimizer	**LR**	**LR Decay**	**Epochs**	**Test Loss**	**Test Acc**
Momentum SGD	10^{-1}	–	6.2	**0.63**	**80.7**
Momentum SGD	10^{-2}	–	4.4	0.63	79.3
Momentum SGD	10^{-3}	–	6.8	0.75	76.12
Momentum SGD	10^{-4}	–	13.6	0.92	68.07
Momentum SGD	10^{-1}	10^{-3}	5.4	0.58	81
Momentum SGD	10^{-1}	10^{-4}	6.4	**0.60**	**81.5**
Momentum SGD	10^{-1}	5.10^{-4}	6.0	0.60	81.44
Adam	10^{-1}	–	9	0.76	77.04
Adam	10^{-2}	–	5.6	0.61	80.17
Adam	10^{-3}	–	6.4	**0.59**	**82.27**
Adam	10^{-4}	–	3.4	0.73	75.35

We performed the naive versions of micro cross-validation experiments on training step verification (i.e. MCV-discard) that only decides to accept or discard training step. To compare MCV method and baseline, epochs that have approximately equal number of updates are selecting as stopped epoch of MCV-discard method. We also performed naive learning rate search algorithm (i.e MCV-lr-voting) which is explained in Algorithm 4 in Chapter 3. Basically, this method combines a naive learning rate search and training verification using MCV. Pre-defined epoch count is used for MCV-lr-voting method. Querying learning rates are $\{0.1, 0.01, 0.001, 0.0001\}$ in this learning rate search method.

Final proposed adaptive learning rate search methods are summarized in following:

- **Adaptive LR search method 1:** Top learning rate is selected based on average loss of validation mini-batches at the end of each epoch. In each step, validation losses are collected then, these losses of each learning rate are averaged at the end of epoch. Learning rate having smallest average validation loss is selected for the subsequent epoch.
- **Adaptive LR search method 2** Top learning rate is selected based on average validation loss and training loss combination at the end of each epoch. In each step, validation and training losses are collected then, these losses of learning rates are averaged at the end of epoch with following formula:
  $(1 - \alpha) * val\_loss + alpha * train\_losses, \alpha \in [0.1, 0.3, 0.5]$
  Learning rate having smallest average loss is selected for the subsequent epoch.
- **Adaptive LR search method 3** This method only uses validation losses of validation mini-batch like method 1. Different from method 1, this approach selects the largest learning rate that decreases the validation loss compared to previous validation loss calculated at the beginning of training step. If none of learning rates decreases validation loss with the training update, learning rate that have minimum validation loss is selected.

For these methods, initial learning rate list, i.e. querying learning rates, is $\{0.1, 0.01, 0.001, 0.0001\}$ and change rule of learning rate list is:
$lr\_list = \left[5lr, 2lr, \frac{4}{3}lr, lr, \frac{3}{4}lr, \frac{1}{2}lr, \frac{1}{5}lr\right]$

Table 4.4: **CIFAR-10 average baseline results on VGG-11 architecture.** Baseline experiments with average epoch count, test accuracy and test loss that are conducted to obtain learning rate and learning rate decay setting of best accuracy result. The same early stopping algorithm is used of other networks. Each setting of learning rate and learning rate decay is repeated 5 times due to randomness of %80-%20 training set, validation set splits for early stopping. The best resulting learning rate and learning rate decay will be used in further experiments of the same VGG-11 architecture. Since none of regularization methods are not applied to network, overfitting starts quickly according to early stopping with 20 epoch patience factor. Reported epoch numbers in the table shows epochs that gives smallest validation error with a 20 more epoch training tries i.e. 20 epoch patience of early stopping.

Optimizer	**LR**	**LR Decay**	**Epochs**	**Test Loss**	**Test Acc**
Momentum SGD	10^{-1}	–	None	None	10
Momentum SGD	10^{-2}	–	9.8	**0.80**	**75.4**
Momentum SGD	10^{-3}	–	42.4	0.96	73.26
Momentum SGD	10^{-4}	–	80	1.77	31.3
Momentum SGD	10^{-2}	10^{-3}	13.2	0.82	73.3
Momentum SGD	10^{-2}	10^{-4}	9.8	**0.78**	**75.72**
Momentum SGD	10^{-2}	$5.\ 10^{-4}$	11	0.85	73.61
Adam	10^{-1}	–	11.2	2.30	10
Adam	10^{-2}	–	12.8	2.30	10
Adam	10^{-3}	–	7.4	**0.84**	**75.1**
Adam	10^{-4}	–	9.2	0.83	73.05

Each of our customs methods is run 10 times to report average results due to randomness of PyTorch Python library and CUDA and random split of mini-batches in MCV validation. Our test environment includes 2x Xeon Scalable 6148 2.40 GHz CPU processor with 16GB RAM and 4x NVIDIA Tesla V100 16GB. Also, Cuda 9.0 version installed on test environment. The models are implemented in Pytorch but we also used Keras implementation with Tensorflow backend in very earlier experiments. Complete results of MCV methods are shown in Table 4.5 that includes small CNN, Resnet-18 and VGG-11 architectures' results.

MCV-discard method's results show that verification of training step and discarding some of them mostly finalized in high accuracy. However, a few outliers of Resnet-18 runs show that training can be failed due to discarding too much update. Moreover, defining a stopping criteria based on number of discarded step count is not possible since number of discard count among epochs is not decreasing or increasing over epochs. MCV-lr-voting combines MCV-discard and naive learning rate search which is explained in Algorithm 4 and 5. Compared to MCV-discard, number of discards are decreased since training tends to select smallest learning rate instead of discarding the step of pre-defined learning rate. Also, MCV-lr-voting results show that some steps select highest learning rates 0.1 and 0.01 at earlier three epochs. However, training continues with selecting more and more smallest learning rates at very earlier epochs that causes underfitting of network and slow down the training. Since MCV-discard and MCV-lr-voting did not yield a consistent improvement on CIFAR-10 (in addition to the other mentioned problems), we decide to continue with other adaptive learning rate search methods (which are in Table 4.5 as Method1, Method2 and Method3) for further experiments of SVHN and Adience.

Table 4.5: **Results of MCV methods on CIFAR-10.** Small CNN, Resnet-18 and VGG-11 test accuracy and loss values are reported with theoretical and time costs.

	Method	**Epoch**	**Test Loss**	**Test Acc.**	**Theoretic Cost**	**Time Cost**
CNN	Baseline SGD	6	0.72	76.6	(~656K ,~378K)	36min 46s
	MCV-discard	44	0.75	77.19	–	–
	MCV-lr-voting	100	1.41	66.74	(~273K,~195K)	25min 40s
	Method 1	53.2	0.91	68.48	(~285K, ~228K)	22min 56s
	Method 2, $\alpha = 0.1$	35.6	1.03	73.59	(~368K, ~173K)	19min 38s
	Method 2, $\alpha = 0.3$	5.2	1.68	38.37	(~167K, ~78K)	8min 54s
	Method 2, $\alpha = 0.5$	4.4	1.49	45.68	(~162K, ~72K)	8min 37s
	Method 3	65.8	2.24	71.96	(~334K, ~268K)	27min 10s
Resnet-18	Baseline SGD	6.2	0.61	81.5	(~480K, ~240K)	2h 2min
	MCV-discard	68	1.51	61.92	–	–
	MCV-lr-voting	100	0.95	70.33	(~273K, ~195K)	1h 17min
	Method 1	100	0.96	65.3	(~390K, ~312K)	1h 40min
	Method 2, $\alpha = 0.1$	100	0.95	68.71	(~664K, ~312K)	2h 12min
	Method 2, $\alpha = 0.3$	55.9	1.01	75.29	(~503K, ~237K)	1h 41min
	Method 2, $\alpha = 0.5$	44.4	0.76	81.03	(~427K, ~201K)	1h 25min
	Method 3	82.7	0.93	76.91	(~390K, ~312K)	1h 40min
VGG-11	Baseline SGD	9.8	0.80	75.72	(~416K,~208K)	1h 34min
	MCV-discard	100	0.89	74.09	–	–
	MCV-lr-voting	100	1.56	41.56	(~137K, ~98K)	33min 30s
	Method 1	70.9	1.04	63.31	(~178K, ~142K)	37min 7s
	Method 2, $\alpha = 0.1$	100	1.87	68.05	(~332K, ~156K)	52min 30s
	Method 2, $\alpha = 0.3$	26	0.89	73.91	(~153K, ~72K)	24min 9s
	Method 2, $\alpha = 0.5$	14.4	0.92	70.74	(~114K, ~54K)	18min 4s
	Method 3	47	1.54	77.51	(~121K, ~105K)	27min 21s

4.6 SVHN Experiments

Comprehensive experiments that measure the accuracy and performance of adaptive learning rate search methods are also conducted for SVHN dataset. Since image dimensions (32x32) are equals to CIFAR-10 dataset images, the same architectures are used as Resnet-18 and VGG-11. However, we used different smaller CNN from CIFAR-10 that have less convolutional filters. Basic CNN architecture designed as four 3x3 convolutional layers which have 32, 32, 64, 64 output channels respectively. Convolutions are followed by 2 fully connected layers, 512 and 10 sized.

The same test environment of CIFAR-10 experiments is also used in SVHN experiments. Baseline methods are obtained with the same early stopping procedure that repeated 5 times for each learning rate setting which is explained in Section 4.5. Baseline results of small CNN, Resnet-18 and VGG-11 are reported in Tables 4.6, 4.7 and 4.8.

Table 4.6: **SVHN average baseline results on small CNN.** Experiments with average epoch count, test accuracy and test loss are reported. They are conducted to obtain learning rate and learning rate decay setting that gives best accuracy result.Each setting of learning rate and learning rate decay is repeated 5 times due to randomness of %80-%20 training set and validation set splits for early stopping. Final accuracy results are obtained from models trained with %100 training set with epoch counts found by early stopping.

Optimizer	**LR**	**LR Decay**	**Epochs**	**Test Loss**	**Test Acc**
Momentum SGD	10^{-1}	–	5	0.55	84.75
Momentum SGD	10^{-2}	–	6	**0.36**	**90.16**
Momentum SGD	10^{-3}	–	16.8	0.46	87.79
Momentum SGD	10^{-4}	–	30	0.97	77.84
Momentum SGD	10^{-2}	10^{-3}	5.6	0.39	89.16
Momentum SGD	10^{-2}	5.10^{-4}	5	0.37	89.56
Momentum SGD	10^{-2}	10^{-4}	4.5	0.36	89.66
Adam	10^{-1}	–	14.2	2.23	19.58
Adam	10^{-2}	–	15.2	2.22	19.58
Adam	10^{-3}	–	3.8	**0.33**	**91.08**
Adam	10^{-4}	–	10	0.41	89.1

Table 4.7: **SVHN average baseline results on Resnet-18.** Experiments with average epoch count, test accuracy and test loss are reported. They are conducted to obtain learning rate and learning rate decay setting that gives best accuracy result.Each setting of learning rate and learning rate decay is repeated 5 times due to randomness of %80-%20 training set and validation set splits for early stopping. Final accuracy results are obtained from models trained with %100 training set with epoch counts found by early stopping.

Optimizer	**LR**	**LR Decay**	**Epochs**	**Test Loss**	**Test Acc**
Momentum SGD	10^{-1}	–	4.2	**0.21**	**94.17**
Momentum SGD	10^{-2}	–	3.4	0.22	93.69
Momentum SGD	10^{-3}	–	6	0.26	92.50
Momentum SGD	10^{-4}	–	23.6	0.30	91.02
Momentum SGD	10^{-1}	10^{-3}	3.8	0.20	94.25
Momentum SGD	10^{-1}	5.10^{-3}	3.2	**0.20**	**94.30**
Momentum SGD	10^{-1}	10^{-4}	3.8	0.21	94.01
Adam	10^{-1}	–	13.4	1.07	62.81
Adam	10^{-2}	–	15.2	0.22	93.63
Adam	10^{-3}	–	3.8	**0.21**	**94.12**
Adam	10^{-4}	–	10	0.26	92.18

Table 4.8: **SVHN average baseline results on VGG-11.** Experiments with average epoch count, test accuracy and test loss are reported. They are conducted to obtain learning rate and learning rate decay setting that gives best accuracy result.Each setting of learning rate and learning rate decay is repeated 5 times due to randomness of %80-%20 training set and validation set splits for early stopping. Final accuracy results are obtained from models trained with %100 training set with epoch counts found by early stopping.

Optimizer	**LR**	**LR Decay**	**Epochs**	**Test Loss**	**Test Acc**
Momentum SGD	10^{-1}	–	None	None	None
Momentum SGD	10^{-2}	–	8	**0.27**	**92.3**
Momentum SGD	10^{-3}	–	37.8	0.26	90.34
Momentum SGD	10^{-2}	10^{-3}	15.8	0.32	91.32
Momentum SGD	10^{-2}	5.10^{-4}	8.6	0.30	91.94
Momentum SGD	10^{-2}	10^{-4}	9.4	**0.30**	**92.53**
Adam	10^{-1}	–	10	2.23	19.58
Adam	10^{-2}	–	11.2	2.22	19.58
Adam	10^{-3}	–	5.4	**0.28**	**92.3**
Adam	10^{-4}	–	7	0.29	91.7

Table 4.9: Adaptive learning rate search results of SVHN. Small CNN, Resnet-18 and VGG-11 architectures' test accuracy and loss values are reported.

	Method	**Epoch**	**Test Loss**	**Test Acc.**	**Theoretic Cost**	**Time Cost**
CNN	Baseline SGD	4.8	0.36	90.16	(~440K, ~220K)	36min 30s
	Method 1	27.8	0.37	89.55	(~137K, ~109K)	14min 25s
	Method 2, $\alpha = 0.1$	33.4	0.62	89.15	(~260K, ~122K)	17min 37s
	Method 2, $\alpha = 0.3$	6	0.55	83.30	(~126K, ~60K)	8min 35s
	Method 2, $\alpha = 0.5$	3.6	1.20	58.85	(~115K, ~54K)	7min 47s
	Method 3	50	0.82	90.27	(~200K, ~160K)	21min 17s
Resnet-18	Baseline SGD	3.2	0.20	94.3	(~388K, ~194K)	2h 47 min
	Method 1	80.5	0.40	88.53	(~286K, ~229K)	2h 2 min
	Method 2, $\alpha = 0.1$	92.6	0.31	91.45	(~496K, ~229K)	2h 53 min
	Method 2, $\alpha = 0.3$	66.25	0.38	93.78	(~420K, ~197K)	2h 29min
	Method 2, $\alpha = 0.5$	23.4	0.27	92.73	(~211K, ~99K)	1h 15min
	Method 3	72.4	0.25	93.95	(~264K, ~212K)	1h 53min
VGG-11	Baseline SGD	9.8	0.30	92.53	(~436K, ~218K)	1h 37min
	Method 1	100	0.45	86.25	(~286K, ~229K)	59min 55s
	Method 2, $\alpha = 0.1$	13.7	1.46	48.7	(~163K, ~77K)	25min 54s
	Method 2, $\alpha = 0.3$	3.4	2.05	26.3	(~114K, ~54K)	17min 59s
	Method 2, $\alpha = 0.5$	2	2.23	19.6	(~107K, ~50K)	16min 54s
	Method 3	45.7	1.01	72.08	(~188K, ~150K)	39min 32s

4.7 Adience Experiments

Since Adience images are collected from real world social media, this dataset represents complex nature. We aim to observe proposed methods performance on this dataset because of this complex nature and providing a different task (rather than CIFAR-10 and SVHN datasets) which is age classification of human faces. Since the age classification task and images are more complex, we used Resnet-50 architecture for Adience experiments The same test environment of previous experiments is also used in Adience experiments. Baseline methods results are obtained with the early stopping procedure that repeated 3 times for each learning rate setting. Since train, test, validation sets are provided, we used provided validation set in all experiments instead of random validation set for each experiment. Another reason of using provided train-test-validation splits is that simple stratified sampling is not enough for this dataset. Because different images of the same subject should not be in both train and test splits in order to prevent overfitting.

Table 4.10: **Adience average baseline results on Resnet-50.** Experiments with average epoch count, test accuracy and test loss are reported. They are conducted to obtain learning rate and learning rate decay setting that gives best accuracy result.Each setting of learning rate and learning rate decay is repeated 5 times.

Optimizer	**LR**	**LR Decay**	**Epochs**	**Test Loss**	**Test Acc**
Momentum SGD	10^{-1}	–	26.6	1.47	50.12
Momentum SGD	10^{-2}	–	12.8	1.65	48.64
Momentum SGD	10^{-3}	–	16.4	1.68	44.81
Momentum SGD	10^{-4}	–	77	1.71	42.61
Momentum SGD	10^{-1}	10^{-2}	139.5	1.44	48.64
Momentum SGD	10^{-1}	10^{-3}	45.6	1.46	49.53
Momentum SGD	10^{-1}	5x10^{-3}	65.4	1.54	45.86
Momentum SGD	10^{-1}	10^{-4}	24.4	**1.37**	**51.02**
Adam	10^{-1}	–	25.2	1.51	47.45
Adam	10^{-2}	–	15.6	1.38	51.94
Adam	10^{-3}	–	11.2	**1.47**	**53.33**
Adam	10^{-4}	–	8	1.42	49.25

Table 4.11: **Adience average adaptive learning rate search using MCV results on Resnet-50.** Experiments with average epoch count, test accuracy and test loss, theoretical and wall clock time cost are reported.

Method	**Epoch**	**Test Loss**	**Test Acc.**	**Theoretic Cost**	**Time Cost**
Baseline SGD	24.4	1.37	51.02	(~522K, ~261K)	26h 16min
Method 1	75.6	1.80	39.11	(~176K, ~141K)	3h 38min
Method 2, $\alpha = 0.1$	96.7	1.63	46.75	(~314K, ~148K)	4h 13min
Method 2, $\alpha = 0.3$	81.6	2.14	48.2	(~314K, ~148K)	4h 13min
Method 2, $\alpha = 0.5$	**72.6**	**2.28**	**52.50**	**(~291K, ~137K)**	3h 54min
Method 3	100	1.48	46.41	(~185K, ~148K)	3h 49min

CHAPTER 5

CONCLUSION

In this thesis, we propose micro cross-validation based methods for deep neural network training. Experiments are conducted to show effects of proposed MCV methods on three dataset (CIFAR-10, SVHN and Adience) and three convolutional neural networks which are small custom CNN, ResNet and VGG architectures. Questions (Q1, Q2 and Q3) posed in Introduction are discussed using the experimental results. **Q1** is questioning the success of MCV-discard which is training step validation using MCV with a given learning rate setting. CIFAR-10 results (Table 4.5) give us a mixed answer. There is an improvement on small custom CNN while other architectures do not have any improvement in terms of test accuracy. Unlike small CNN, Resnet-18 experiments include erroneous runs because, discarding too many training steps causes to stuck training on very poor state. Another question, **Q2**, that we explored in this work is the convenience of MCV for automated learning rate search. For this reason, we proposed two different adaptive learning rate search methods. The first one computes a moving average of the votes for the best batch-level learning rates, and instantaneously uses the highest-voted learning rate for actual gradient updates. In the second method, a constant LR is used during an epoch and while doing so, average validation loss is computed per LR in the search array. At the beginning of a new epoch, we choose the best. For this second method, we proposed a set of different loss functions. Finally, as an answer to **Q3**, we analyzed the improvements of automated learning rate search methods with respect to baseline. MCV based learning rate search methods do not provide consistent improvement in our experiments. However, reasonable accuracy is obtained in a much less amount of time compared to the classical macro-CV based grid search.

REFERENCES

[1] Y. Huang, Y. Cheng, D. Chen, H. Lee, J. Ngiam, Q. V. Le, and Z. Chen, "GPipe: Efficient Training of Giant Neural Networks using Pipeline Parallelism," 2018.

[2] J. Hu, L. Shen, and G. Sun, "Squeeze-and-Excitation Networks," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 6 2018.

[3] M. Tan and Q. Le, "{E}fficient{N}et: Rethinking Model Scaling for Convolutional Neural Networks," in *Proceedings of the 36th International Conference on Machine Learning* (K. Chaudhuri and R. Salakhutdinov, eds.), vol. 97 of *Proceedings of Machine Learning Research*, (Long Beach, California, USA), pp. 6105–6114, PMLR, 2019.

[4] K. Chen, J. Pang, J. Wang, Y. Xiong, X. Li, S. Sun, W. Feng, Z. Liu, J. Shi, W. Ouyang, C. C. Loy, and D. Lin, "Hybrid Task Cascade for Instance Segmentation," 2019.

[5] B. Singh, M. Najibi, and L. S. Davis, "SNIPER: Efficient Multi-Scale Training," in *Advances in Neural Information Processing Systems 31* (S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, eds.), pp. 9310–9320, Curran Associates, Inc., 2018.

[6] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 6 2016.

[7] F. Schroff, D. Kalenichenko, and J. Philbin, "FaceNet: A unified embedding for face recognition and clustering," *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 6 2015.

[8] Q. Cao, L. Shen, W. Xie, O. M. Parkhi, and A. Zisserman, "VGGFace2: A Dataset for Recognising Faces across Pose and Age," *2018 13th IEEE Inter-*

national Conference on Automatic Face & Gesture Recognition (FG 2018), 5 2018.

[9] S. E. Wei, V. Ramakrishna, T. Kanade, and Y. Sheikh, “Convolutional pose machines,” *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 2016-Decem, pp. 4724–4732, 2016.

[10] W. Yang, S. Li, W. Ouyang, H. Li, and X. Wang, “Learning Feature Pyramids for Human Pose Estimation,” pp. 1290–1299, 2017.

[11] G. E. Hinton, N. Srivastava, and K. Swersky, “Neural Networks for Machine Learning,” *COURSERA: Neural Networks for Machine Learning*, 2012.

[12] D. P. Kingma and J. L. Ba, “Adam: A method for stochastic gradient descent,” in *ICLR: International Conference on Learning Representations*, 2015.

[13] A. C. Wilson, R. Roelofs, M. Stern, N. Srebro, and B. Recht, “The Marginal Value of Adaptive Gradient Methods in Machine Learning,” in *Advances in Neural Information Processing Systems 30* (I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, eds.), pp. 4148–4158, Curran Associates, Inc., 2017.

[14] J. Zhang, I. Mitliagkas, and C. Ré, “YellowFin and the Art of Momentum Tuning,” *CoRR*, vol. abs/1706.0, 2017.

[15] B. T. Polyak, “Some methods of speeding up the convergence of iteration methods,” *USSR Computational Mathematics and Mathematical Physics*, vol. 4, no. 5, pp. 1–17, 1964.

[16] J. Duchi, E. Hazan, and Y. Singer, “Adaptive subgradient methods for online learning and stochastic optimization,” *The Journal of Machine Learning Research*, vol. 12, pp. 2121–2159, 2011.

[17] N. S. Keskar and R. Socher, “Improving Generalization Performance by Switching from Adam to SGD,” 2017.

[18] M. Zaheer, S. Reddi, D. Sachan, S. Kale, and S. Kumar, “Adaptive Methods for Nonconvex Optimization,” in *Advances in Neural Information Processing Sys-*

tems 31 (S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, eds.), pp. 9793–9803, Curran Associates, Inc., 2018.

[19] J. Chen and Q. Gu, "Closing the Generalization Gap of Adaptive Gradient Methods in Training Deep Neural Networks," *CoRR*, vol. abs/1806.0, 2018.

[20] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," *CVPR*, 2015.

[21] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, 2017.

[22] S. Jenni and P. Favaro, "Deep Bilevel Learning," 2018.

[23] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.

[24] Y. Bengio, "Practical recommendations for gradient-based training of deep architectures," in *Neural networks: Tricks of the trade*, pp. 437–478, Springer, 2012.

[25] L. Bottou, "Stochastic Gradient Descent Tricks," in *Neural Networks: Tricks of the Trade*, pp. 421–436, Springer, 2012.

[26] S. L. Smith, P.-J. Kindermans, C. Ying, and Q. V. Le, "Don't Decay the Learning Rate, Increase the Batch Size," in *ICLR: International Conference on Learning Representations*, 2018.

[27] J. Bergstra and Y. Bengio, "Random Search for Hyper-Parameter Optimization," *Journal of Machine Learning Research*, vol. 13, pp. 281–305, 2012.

[28] L. N. Smith, "Cyclical Learning Rates for Training Neural Networks," *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*, 3 2017.

[29] L. N. Smith and N. Topin, "Super-Convergence: Very Fast Training of Residual Networks Using Large Learning Rates," *CoRR*, vol. abs/1708.0, 2017.

[30] I. Loshchilov and F. Hutter, "SGDR: Stochastic Gradient Descent with Warm Restarts," in *ICLR: International Conference on Learning Representations*, pp. 1–16, 2017.

[31] T. Schaul, Z. Sixin, and Y. LeCun, “No More Pesky Learning Rates,” in *Proceedings of the 30th International Conference on Machine Learning*, 2013.

[32] A. G. Baydin, R. Cornish, D. M. Rubio, M. Schmidt, and F. Wood, “Online Learning Rate Adaptation with Hypergradient Descent,” in *International Conference on Learning Representations*, 2018.

[33] Y. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller, “Efficient BackProp,” in *Neural Networks: Tricks of the Trade, This Book is an Outgrowth of a 1996 NIPS Workshop*, (London, UK), pp. 9–50, Springer-Verlag, 1998.

[34] A. Krizhevsky, “Learning Multiple Layers of Features from Tiny Images,” tech. rep., 2009.

[35] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y Ng, “Reading Digits in Natural Images with Unsupervised Feature Learning,” *NIPS*, 2011.

[36] E. Eidinger, R. Enbar, and T. Hassner, “Age and Gender Estimation of Unfiltered Faces,” *IEEE Transactions on Information Forensics and Security*, vol. 9, pp. 2170–2179, 12 2014.

[37] K. Simonyan and A. Zisserman, “Very Deep Convolutional Networks for Large-Scale Image Recognition,” in *ICLR: International Conference on Learning Representations*, pp. 1–14, 2015.