SAKARYA UNIVERSITY JOURNAL OF COMPUTER AND INFORMATION SCIENCES
VOL. 4, NO. 3, DECEMBER 2021
DOI: 10.35377/saucis.01.02.935353

Research Article

Automated learning rate search using batch-level cross-validation

Duygu Kabakcı[1], Emre Akbaş[2]
[1]Middle East Technical University; duygukabakci@gmail.com
[2]Corresponding Author; Middle East Technical University; emre@ceng.metu.edu.tr

Received 10 May 2021; Revised 02 November 2021; Accepted 04 November 2021; Published online 31 December 2021

Abstract

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 network architecture 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 (CV), 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. In this paper, we explore batch-level cross-validation as an alternative to the classical dataset-level, hence macro, CV. The advantage of batch-level or micro CV methods is that the gradient computed during training is re-used to evaluate several different learning rates. We propose an algorithm based on micro CV and stochastic gradient descent with momentum, which produces a learning rate schedule during training by selecting a learning rate per epoch, automatically. In our algorithm, a random half of the current batch (of examples) is used for training and the other half is used for validating several different step sizes or learning rates. We conducted comprehensive experiments on three datasets (CIFAR10, SVHN and Adience) using three different network architectures (a custom CNN, ResNet and VGG) to compare the performances of our micro-CV algorithm and the widely used stochastic gradient descent with momentum in an early-stopping macro-CV setup. The results show that, our micro-CV algorithm achieves comparable test accuracy to macro-CV with a much lower computational cost.

Keywords: deep learning, neural networks, learning rate, hyper-parameter search, adaptive learning rate, cross-validation

1. Introduction

Training deep neural network models is not a trivial task. It is a delicate and complex process which aims to efficiently obtain a model that generalizes well to unseen test data. An excessive number of hyper-parameters including learning rate, learning rate schedule, mini-batch size, regularization parameter, weight decay constant and the network architecture increase the complexity of the training process as all of these parameters need to be tuned. The problem is exacerbated for new problem domains or new datasets. 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. In fact, human skill and expertise is a significant factor in the final performance of a deep learning architecture [1]. Even carefully designed sequential experiments conducted with knowledge-based assessments come with a very high amount of computational cost. Given the surprisingly large carbon-footprint of deep learning computations [2], electricity and hardware costs, efforts to shorten the training process become crucial.

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, the learning rate adjusts the magnitude of the network's weight updates for minimizing the loss function. If the learning rate is too high, the model struggles to converge to a local minima; while a too small learning rate slows down convergence, hence increases the total training time.

Adaptive optimizers (e.g. Rmsprop[3], Adam[4]) have been proposed to address the problem of setting optimal learning rates. It is widely reported that "Adam" with its default parameters achieves high accuracy for many architectures. However, there is also evidence that "researchers kept evolving new architectures on which Adam works" [5]. Recent research ([6, 7]) also points out the convergence problem of Adam optimizer. According to these recent findings, hand-tuned stochastic gradient descent (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.

The standard way to find an optimal learning rate and/or decay parameter (more generally, a learning rate schedule) is to apply cross-validation (CV) to estimate model performance on unseen data for different learning rates and decay values. In classical cross-validation (CV), a random part of the dataset is reserved for the evaluation of model to estimate its future performance on unseen data. This process is repeated multiple times with random validation sets to determine the best learning rate settings. We name this standard CV method as "macro CV" since it is a dataset-level method. As an alternative, we propose "micro CV" method which works at the mini-batch level. We argue that "micro CV" may be more efficient because once the gradient vector is computed at the batch level, it could be re-used to test out several different learning rates. We propose an automated learning rate selection algorithm that aims to find optimal learning rate and learning rate schedules during training. In each iteration of our algorithm, we sample a mini-batch from the training set just like in all stochastic gradient algorithms. Then, we split this batch into two equal-size sets, one of which is used as the training mini-batch and the other as the validation mini-batch. We compute the gradient of the loss function with respect to network weights using only the "training" mini-batch. Once we have the gradient vector, we apply different step-sizes or learning rates along this vector to obtain different future models. Next, we evaluate these future models on the "validation" mini-batch, and accumulate the corresponding losses. When this process is repeated for the whole training set (i.e. all batches), we identify the step-size which has accumulated the least amount of (validation) loss, and set it as the learning rate for the next epoch. We implemented this method and a variant of it, where we not only accumulate the validation loss but also the training loss, and used them for the image classification task on three different image classification datasets (CIFAR10, SVHN, Adience) with three different CNN architectures. Our experiments show that, in terms of test set classification accuracy, macro-CV slightly outpeforms micro-CV, however, in terms of computational cost, micro-CV algorihtm is better by a large margin. Overall, our micro-CV algorithm yields promising results.

This paper has been compiled from the first author's MSc thesis [8]. The rest of the paper is organized as follows. Section 2 discusses the SGD algorithm and provides a comparison with adaptive gradient descent algorithms as background. Section 3 describes our proposed micro CV algorithm. In Section 4, we describe the experiments we conducted for the purpose of comparing micro-CV and macro-CV methods. Finally, Section 5 concludes the paper by providing a brief summary and discussion.

2. Background and Related Work

Many methods have been proposed for automated learning rate selection [3, 4, 7, 9, 10, 11, 12, 13, 14, 15]. The ultimate goal is to obtain the lowest generalization error in a minimum time and memory budget. This section covers traditional hyper-parameter tuning approaches, adaptive learning rate methods, cyclical learning rate schedules, gradient based learning rate tuning methodologies and mini-batch validation based methods.

2.1 Hyper-parameter Tuning and Cross Validation

Automated hyper-parameter tuning can be considered as the ancestor of automated learning rate tuning. The classical way of tuning is to define a set of parameter values and estimate model performance for them. This basic method is known as "grid search" and it can be carried out in a sequential or parallel

manner, for more than one variable or hyper-parameter. The set of parameter values can be set beforehand or sampled randomly as well (i.e., random search [10]). The model performance is evaluated on a held-out set known as the "validation set". Typically, $10 - 30\%$ of the training set is reserved for this purpose. To reduce variability of results, the training set can be split into many folds and each fold can be used as the validation set while others are used for training. These methods are generally known as "cross-validation" methods. In this paper, we further call these methods as "macro cross-validation" since the train-validation split is done at the dataset level. As an alternative, this split can be carried out at the batch-level as we propose in this paper.

2.2 Adaptive Learning Rate Methods

The earliest heuristic to achieve adaptive learning is the idea of using separate learning rates per parameter based on the sensitivity of the cost function per parameter. AdaGrad [9], an earlier example of adaptive learning rate optimizers, scales learning rate per parameter with the square root of the sum of all historical squared values of the gradient. In this approach, parameters with larger partial derivatives have decreased learning rates compared to parameters with small partial derivatives. Even though this approach helps some models, keeping historical partial derivatives for the whole training session may lead to excessive decrease in the effective learning rate for some parameters. Rmsprop [3] modifies the AdaGrad algorithm to address its excessive decrease on learning rate by using a moving average of gradients to replace the whole-history based average. This introduces a new hyperparameter (which is usually not tuned). Adam optimizer [4] combines Rmsprop's and momentum optimizer's benefits. It computes two historical moving average estimates which keep averages of gradients and squared 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 tuning the global learning rate and other hyper-parameters.

2.2.1 Revisiting SGD with Momentum

Even though adaptive methods are quite useful, recent research shows that SGD with momentum can obtain better test results over adaptive methods. Wilson et al. [6] conduct a comprehensive study which shows that, contrary to common belief, SGD and SGD with Momentum outperform adaptive optimizers on unseen dataset in designed tasks for over-parameterized models. 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 brings considerable improvement compared to using the default settings. Another research inspired by Wilson et al.'s [6] findings suggests a simple idea: using Adam on earlier epochs of deep neural network training, then switching to SGD with Momentum to address the saturation problem of Adam on later epochs [16]. They showed that using Adam and SGD with momentum instead of only using Adam results in better generalization. Several modifications have also been proposed for the Adam optimizer. The "YOGI" algorithm [17] proposes an additive adaptive rule which controls the increase in learning rate, as opposed to the rapid increase yield by Adam. Zhang et al. [7] conduct several experiments that compares hand tuned "SGD with momentum" and adaptive optimization methods. They report that models trained with hand-tuned 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, use the "SGD with momentum" method [18] [19] [20] [21].

2.2.2 Cyclical Learning Rates

Cyclical learning rates (CLR) method [11] addresses the learning rate and learning rate schedule tuning problem. The method trains the neural network by cyclically varying learning rates within predefined boundaries instead of always 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, the learning rates where model

accuracy starts to increase and decrease are taken as minimum and maximum boundaries of the cycle. Learning rate varies between these minimum to maximum and maximum to minimum boundary in a defined step size during training. These method also requires the selection of step size, i.e the 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. In a follow-up research, Smith et al. [12] propose super convergence phenomenon that can be achieved by using very large learning rates in the 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 was only demonstrated in a single task which limits the generalizability of this method.

Another adaptive method similar to cyclical learning rates is "SGD with warm restarts" (SGDR) [13]. In this work, the learning rate is initialized 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 that increasing the learning rate from time to time to reasonably higher values can be beneficial for final network performance.

2.2.3 Gradient Based Tuning Methods

Another adaptive approach is using gradients to find out the optimal learning rate. Schaul et al. [14] define a formula to obtain the optimal learning rates for SGD based on the variance of the gradients. This method can find either single global learning rate, or learning rates for each parameter or parameter group, based on the moving averages of gradients and the diagonal Hessian. This algorithm automatically decreases learning rate to zero when loss function is approaching to its optimal value without any manual learning rate search. Zhang. et al. [7] conduct experiments to show that carefully hand-tuned learning rate can be competitive with adaptive learning rate optimizers. They not only analyzed this phenomena but also, came up with an automated learning rate and momentum tuning approach called Yellowfin. They consider the learning rate tuning problem together with momentum tuning. Similar to Schaul et al.'s work [14], 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 the learning rate [15]. 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. Applying hypergradient descent on SGD, SGD with Nesterov momentum and Adam has showed that the need for manual tuning is reduced.

2.2.4 Mini-batch validation

We are not the first to explore mini-batch-level validation. Recently, Jenni and Favaro have extended the idea of cross-validation to validate a group of mini-batches with another mini-batch during training [22]. In their method, called "Deep Bilevel Learning" (DBL), at each iteration, they sample a number of training mini-batches and a validation mini-batch. Then, they calculate the gradient of the loss function on all of these mini-batches. Next, the inner product between the gradients of a training mini-batch and the validation mini-batch is computed. Each training gradient is weighted by its corresponding inner product value with the validation mini-batch. The final training gradient to be used for this iteration is the linear combination of all training mini-batch gradients weighted by their inner products with the validation mini-batch. If the gradient of a training mini-batch agrees with the gradient of the validatio mini-batch, then their inner product, hence the training gradient's weight, is a large positive value. They show that this procedure results in better generalization. They explain this achievement as the validation

of gradients helps to avoid memorization by encouraging model parameter updates that only reduce errors on shared sample patterns. Our proposed methods are similar to DBL in the sense that both use training and validation mini-batches during training. However, while DBL is based on gradient similarity (based on inner product), our methods are based on loss values obtained on training and validation mini-batches.

3. Automated Learning Rate Search Methods using Micro Cross Validation

In this section, we describe our proposed micro-CV based automated learning rate search algorithm. The training scenario we consider is a typical one: we are given a supervised training set $T = \{(x_i, y_i)\}_{i=1}^{n}$ consisting of *n* examples, a neural network architecture represented by $f(x;\theta)$, where $f(\cdot)$ is the overall function computed by the network, θ is the set of learnable weights of the network and *x* is an input to the network, and a loss function $\mathcal{L}$. Our goal is to minimize $\mathcal{L}$ on *T* by adjusting the network weights θ:

$$\theta^* = arg\ \min_{\theta} \quad \frac{1}{n}\sum_{(x_i,y_i)\in T} \mathcal{L}(f(x_i;\theta), y_i). \tag{1}$$

We consider the stochastic gradient descent (SGD) method where a mini-batch *B* consisting of *m* examples is randomly sampled ($m << n$), on which the gradient vector is computed:

$$g = \frac{1}{m}\sum_{(x_i,y_i)\in B} \nabla_\theta \mathcal{L}(f(x_i;\theta), y_i). \tag{2}$$

Then, the learning takes place by updating the network weights θ along the gradient vector g with a step-size η, called the learning rate:

$$\theta^{(t+1)} = \theta^{(t)} - \eta^{(t)} g^{(t)}, \tag{3}$$

where superscript (t) indicates the iteration number. For simplicity, we drop this notation for iteration in the following.

Given the widespread use and its more stable convergence, we consider the "SGD with momentum method" [23, 24], which accelerates learning if recent gradient vectors are consistently aligned. It has an additional parameter α, called the momentum parameter, to decide how much of past gradients are taken into account:

$$\vartheta = \alpha\vartheta - \eta g, \tag{4}$$

$$\theta = \theta + \vartheta. \tag{5}$$

Here the learning rate η is usually a small constant throughout the whole training or a variable which changes its value according to some schedule as a function of time (iteration number). Although decay schedules, where η monotonically decreases through time are more common; cyclical schedules, where η increases and decreases alternatingly [11], are also possible. In classical, macro (i.e. dataset-level) cross-validation (CV), these learning rate schedules are determined prior to training, mostly based on previous experience.

In this paper, we are interested in setting the learning rate η automatically using micro (i.e. batch-level) crossvalidation (CV). Before we describe how we do this, let us first look at the classical, macro CV.

In macro CV, the training set *T* is split into two mutually exclusive, training and validation sets (e.g. 80% to %20 is common). Together with this, a set of learning rate schedules are determined prior to training. Note that a "schedule" deterministically defines the values of η throughout a whole training episode. Then, a complete training episode is carried out for each different learning rate schedule. During training, "early-stopping" technique is widely used as a stopping criterion. Loss on the validation set is monitored and if it does not improve for a certain number of epochs, the training is terminated. At

the end of the training, performance is measured on the validation set and recorded. After all the learning rate schedules are used this way, the one yielding the highest performance (or the lower loss) on the validation set is identified as the "optimal learning rate (schedule)". Many variations to this basic recipe is possible such as using more than one validation set (i.e. k-fold CV).

In contrast to macro CV, in micro CV, there is no need to split T into two prior to training. Instead, each minibatch is split randomly into two at each iteration. On one half-batch, the gradient is computed and on the other half-batch, validation losses for different learning rates are measured. These losses are accumulated for an entire epoch, at the end of which, the learning rate yielding the smallest validation loss is chosen as the learning rate to be used for the next epoch. This way, an "optimal" learning rate is identified for each epoch, which effectively produces a dynamic and adaptive learning rate schedule. Our micro-CV based automated learning rate search method is given in Algorithm 1, which is also illustrated in Figure 1.

Considering the inputs of the classical macro-CV stochastic gradient training algorithm, our micro-CV algorithm has an additional input parameter, λ, which weighs the importance of training and validation losses incurred during training. When λ is 1, only the validation loss is taken into account, and in that case, there is no need to execute line 14 in Algorithm 1.

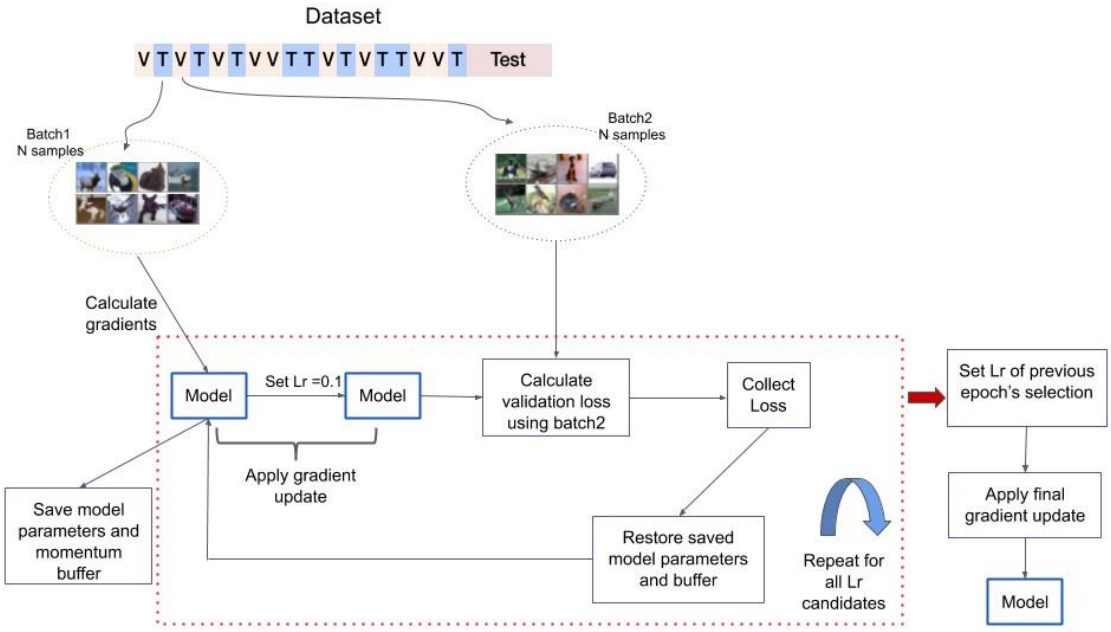

Figure 1 An overview of our micro-CV algorithm. In each training step, a training mini-batch (Batch1) and validation mini-batch (Batch2) are sampled from the training dataset. The gradient of the objective function with respect to the model parameters are computed on Batch1. This gradient is used for "validating" different learning rates on Batch2.

4. Experiments

In this section, we present our experimental results for the micro (i.e. batch-level) cross-validation (MCV) methods described in the previous section, and compare them to the performance of our baseline method. The baseline we consider is the classical macro-CV using stochastic gradient descent with momentum. We test whether our MCV methods that automatically select learning rates improve test set accuracy over the baseline. We also compare our method and the baseline in terms of time complexity.

Algorithm 1 Our "micro-CV" based training algorithm. This algorithm trains a given model by using micro cross-validation to automate the selection of learning rate schedule.

Input: Training set $\boldsymbol{T}$, initial network weights $\boldsymbol{\theta}$, momentum parameter $\boldsymbol{\alpha}$, initial velocity $\boldsymbol{\vartheta}$, mini-batch size $\boldsymbol{m}$, training-vs-validation trade-off parameter $\boldsymbol{\lambda}$.
Output: Weights of trained model, $\boldsymbol{\theta}$.

Initialize learning rate $\eta \leftarrow 0.00001$
epoch $\leftarrow$ **1**
while true **do**
  learning_rates $\leftarrow \left[5\eta, 2\eta, \frac{4}{3}\eta, \eta, \frac{3}{4}\eta, \frac{1}{2}\eta, \frac{1}{5}\eta\right]$ // List of learning rates to search for
  losses $\leftarrow$ [0,0,0,0,0,0,0] // Accumulated loss for each learning rate
  for i = 1 ... $|T|/m$ **do** // Loop over mini-batches
    Sample B, a mini-batch of m examples from T
    Randomly split B into two half-batches, B_1 and B_2 // B_1 is for training, B_2 for validation
    Compute gradient g on B_1: $g \leftarrow \frac{1}{m}\nabla_\theta \sum_{(x_i,y_i)\in B_1} \mathcal{L}\left(f(x_i;\theta), y_i\right)$
    for $j = 1$ ... |learning_rates| **do**
      $\gamma \leftarrow$ learning_rates$_j$ // j^{th} element in learning_rates
      Create a temporary model using learning rate γ: $\theta^{\text{tmp}} \leftarrow \theta + (\alpha\vartheta - \gamma g)$
      Compute the losses on B_1 and B_2 using this temporary model:
      $\ell_{B_1} \leftarrow \frac{1}{m}\sum_{(x_i,y_i)\in B_1} \mathcal{L}\left(f(x_i;\theta^{\text{tmp}}), y_i\right)$ // training loss
      $\ell_{B_2} \leftarrow \frac{1}{m}\sum_{(x_i,y_i)\in B_2} \mathcal{L}\left(f(x_i;\theta^{\text{tmp}}), y_i\right)$ // validation loss
      losses$_j \leftarrow$ losses$_j + (1-\lambda)\ell_{B_1} + \lambda\ell_{B_2}$
    end for
    $\vartheta = \alpha\vartheta - \eta g$
    $\theta \leftarrow \theta + \vartheta$ // The actual training update
  end for
  $k \leftarrow$ argmin (losses)
  $\eta \leftarrow$ learning_rates$_k$ // Pick the best learning rate with minimal accumulated loss
  overall_loss(epoch) $\leftarrow$ losses$_k$
  if no improvement in overall_loss **then**
    stop training
  end if
  epoch $\leftarrow$ epoch +1
end while

Using just one dataset and a neural network architecture to explore whether MCV is effective would give us a limited picture. In order to increase the generality of our results, in our experiments, we used three image classification datasets, two of them being widely used small-scale benchmark datasets (CIFAR-10 [25] and SVHN [26]) and one of them being a larger scale dataset for age prediction from face images (Adience [27]). On these datasets, we evaluated three different convolutional neural network (CNN) architectures: a small, custom CNN, a ResNet [18] and a VGG [28] network.

Below we first describe the datasets (Section 4.1), the network architectures (Section 4.2) and our performance measures (Section 4.3). Then, we present and discuss the results of our experiments.

4.1 Datasets: CIFAR-10, SVHN, Adience

The CIFAR-10 [25] dataset contains 60,000 32x32 color images from 10 classes which are airplane, automobile, bird, cat, deer, dog, frog, horse, ship and truck. The training set contains 50,000 images and the remaining 10,000 images are used as testing images.

The Street View House Numbers (SVHN) dataset [26] 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 73,257 training and 26,032 testing examples. Each image is 32x32 pixels.

The Adience [27] 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 with 8 age interval classes {(0-2, 4-6, 8-13,15-20, 25-32,38-43,48-53,60−)}. We use the cropped and aligned images version of the dataset.

Provided train, test, validation sets include 11823, 4316 and 1284 images respectively. We performed only the age prediction task, skipping gender prediction.

4.2 Network Architectures

On CIFAR-10 and SVHN datasets, we used two well-known and widely used architectures: ResNet-18 [18] and VGG-11 [28], as well as a basic and small CNN, which has only convolutional and fully connected layers without any regularization such as batch normalization and dropout. The small CNN 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 have 512, 265, 10 output channels, respectively. Rectified linear unit (ReLU) [18] is used as nonlinear activation function after each convolutional and fully-connected layer. For the SVHN dataset, since digit classification is an easier task, our small CNN architecture has four 3x3 convolutional layers which have 32, 32, 64, 64 output channels, respectively. Convolutions are followed by 2 fully connected layers with 512 and 10 output channels. The Adience dataset, unlike CIFAR-10 and SVHN, contains high-resolution images and it is a relatively more challenging dataset. A large capacity neural network can be beneficial for this dataset's age prediction task. For this reason, we choose to use ResNet-50 [18] architecture for this dataset.

4.3 Performance Measures

We compare the performances of baseline methods and our automated learning rate search methods in terms of accuracy and time. To compare accuracy, we simply use the ratio (percent) of correctly classified examples in the testing set. To compare time expenditure, we use two different performance measures: wall-clock time and theoretical computational complexity.

The "wall-clock time" is simply the time spent running all the experiments required for training a given model on a given dataset from start to end. Here, by training, we mean the whole cross-validation process.

The "theoretical computational complexity" refers to the number of complex operations (with large time expenditure), namely, the forward propagation, backward propagation and weight update operations. Forward operation refers to calculation of output layer's values through passing all neurons of DNN with input. A single forward is required to calculate the loss incurred by the prediction of network and the ground-truth class. 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 the update operation. This operation calculates the final parameter update inside the optimizer with gradients (e.g. momentum buffer calculation for SGD with momentum). In the following, we present an experiment's theoretical complexity with two numbers: (i) number of total forward and backward operations, (ii) number of total update operations.

We conducted our experiments on computers that have two Xeon Scalable 6148 2.40 GHz CPU processor with 16GB RAM and 4x NVIDIA Tesla V100 16GB. The models are implemented in PyTorch but we also used Keras with Tensorflow backend in our earlier experiments.

4.4 CIFAR-10 Experiments

Macro-CV. To establish the baseline results, we used the classical macro cross-validation with early stopping. For each architecture, we performed two sets of experiments which are summarized below.

1. Use SGD with momentum (abbreviated as "Momentum SGD") as the optimizer. No learning rate decay. Search for the best learning rate (LR) among {0.1,0.01,0.001,0.0001}.
2. Use "Momentum SGD" as the optimizer. Use the best learning rate from the previous set. Search for the best learning rate decay parameter among $\{10^{-3}, 5 \times 10^{-4}, 10^{-4}\}$. We follow the standard, inverse-time decay rule:

$$\mathrm{LR} \times \frac{1}{1 + \mathrm{decay} \times \mathrm{step}}, \tag{6}$$

where "decay" is the decay parameter and "step" refers to the number of learning updates on the model, or, simply the iteration number.

Both sets of experiments above have a common macro-CV setup. For a given learning rate schedule (i.e. learning rate and learning rate decay parameter), we randomly split the training set into %80 for training and %20 for validation, with stratified sampling. We monitor the validation loss and use early

Table 1 CIFAR-10 baseline macro-CV results for three different CNN architectures. Each row identifies a different training setup, which consists of a learning rate and a learning rate decay parameter. For each training setup, the training set is randomly split into 80%-20% training and validation. Early stopping with a patience of 20 epochs is used. Reported epoch numbers are for the epochs that yield smallest validation error. Optimizer is Momentum SGD. See Section 4.4 for other details.

	LR	**LR Decay**	**Epochs**	**Test Loss**	**Test Acc.**
Small CNN	10^{-1}	–	3	1.77	33.50
	10^{-2}	–	6	**0.72**	**76.37**
	10^{-3}	–	16.6	0.86	72.23
	10^{-4}	–	80	0.97	67.21
	10^{-2}	10^{-3}	7.4	0.77	74.63
	10^{-2}	5×10^{-4}	6.8	0.75	75.46
	10^{-2}	10^{-4}	6.4	**0.72**	**76.60**
ResNet-18	10^{-1}	–	6.2	**0.63**	**80.70**
	10^{-2}	–	4.4	0.63	79.30
	10^{-3}	–	6.8	0.75	76.12
	10^{-4}	–	13.6	0.92	68.07
	10^{-1}	10^{-3}	5.4	0.58	81.00
	10^{-1}	5×10^{-4}	6.4	**0.60**	**81.50**
	10^{-1}	10^{-4}	6.0	0.60	81.44
VGG-11	10^{-1}	–	None	None	10.00
	10^{-2}	–	9.8	**0.80**	**75.40**
	10^{-3}	–	42.4	0.96	73.26
	10^{-4}	–	80	1.77	31.30
	10^{-2}	10^{-3}	13.2	0.82	73.30
	10^{-2}	5×10^{-4}	9.8	**0.78**	**75.72**
	10^{-2}	10^{-4}	11	0.85	73.61

stopping with a patience of 20 epochs. Since the %80-%20 split introduces randomness on results, we repeat the mentioned process for 5 times to obtain average results. These 5 runs provide us with the required epoch count and the best learning ratverye schedule. Then, we train the model one last time using these settings on the whole original training set (no splits). Finally, performance is reported on the testing set.

The results of these experiments are given in Table 1 for the three architectures mentioned in Section 4.2, namely, small CNN, ResNet-18 and VGG-11. For all three architectures, Momentum SGD with decay yields the best results: 76.60% for small CNN, 81.50% for ResNet-18 and 75.72% for VGG-11. Optimal learning rate (LR) and decay parameters are all different for different architectures: LR=10^{-2}, decay=10^{-4} for small CNN; LR=10^{-1}, decay=5×10^{-4} for small ResNet-18; LR=10^{-2}, decay=5×10^{-4} for VGG-11.

Micro CV. We obtained our micro-cross validation results for four different λ values ($\lambda \in \{1,0.9,0.7,0.5\}$). Due to the randomness introduced by splitting each mini-batch into two (train and

validation half-batches), we repeated each experiment 5 times and reported the average epoch count, test loss and accuracy, theoretical time cost and wall-clock times. Results are presented in Table 2.

For all three architectures, macro-CV outperforms our micro-CV algorithm, in terms of test accuracy. However, we also observe that micro-CV performs reasonably well in a much shorter time window. For the small CNN, it achieves 73.59 test accuracy, which is 96% of macro-CV's test accuracy (76.60) in only 23% of the time spent by macro-CV (8min 37s vs. 36min 46s). The situation is similar for the other two architectures: for ResNet-18, 99% of macro-CV's test accuracy is achieved in 70% of time spent by macro-CV; for VGG-11, these numbers are 98% and 15%.

Table 2 Comparison of macro-CV and micro-CV results 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
Small CNN	Macro-CV baseline	6	0.72	**76.60**	(656K, 378K)	36min 46s
	Micro-CV ($\lambda = 1$)	53.2	0.91	68.48	(285K, 228K)	22min 56s
	Micro-CV ($\lambda = 0.9$)	35.6	1.03	73.59	(368K, 173K)	19min 38s
	Micro-CV ($\lambda = 0.7$)	5.2	1.68	38.37	(167K, 78K)	8min 54s
	Micro-CV ($\lambda = 0.5$)	4.4	1.49	45.68	**(162K, 72K)**	**8min 37s**
ResNet-18	Macro-CV baseline	6.2	0.61	**81.50**	(480K, 240K)	2h 2min
	Micro-CV ($\lambda = 1$)	100	0.96	65.30	(390K, 312K)	1h 40min
	Micro-CV ($\lambda = 0.9$)	100	0.95	68.71	(664K, 312K)	2h 12min
	Micro-CV ($\lambda = 0.7$)	55.9	1.01	75.29	(503K, 237K)	1h 41min
	Micro-CV ($\lambda = 0.5$)	44.4	0.76	81.03	**(427K, 201K)**	**1h 25min**
VGG-11	Macro-CV baseline	9.8	0.80	**75.72**	(416K,208K)	1h 34min
	Micro-CV ($\lambda = 1$)	70.9	1.04	63.31	(178K, 142K)	37min 7s
	Micro-CV ($\lambda = 0.9$)	100	1.87	68.05	(332K, 156K)	52min 30s
	Micro-CV ($\lambda = 0.7$)	26	0.89	73.91	(153K, 72K)	24min 9s
	Micro-CV ($\lambda = 0.5$)	14.4	0.92	70.74	**(114K, 54K)**	**18min 4s**

Table 3 SVHN baseline (macro-CV) results for three different CNN architectures. Each row identifies a different training setup, which consists of a learning rate and learning rate decay parameter. For each training setup, the training set is randomly split into 80%-20% training and validation. Early stopping with a patience of 20 epochs is used. Reported epoch numbers are for the epochs that yield smallest validation error. Optimizer is Momentum SGD. See Section 4.4 for other details.

	LR	LR Decay	Epochs	Test Loss	Test Acc.
Small CNN	10^{-1}	–	5	0.55	84.75
	10^{-2}	–	6	**0.36**	**90.16**
	10^{-3}	–	16.8	0.46	87.79
	10^{-4}	–	30	0.97	77.84
	10^{-2}	10^{-3}	5.6	0.39	89.16
	10^{-2}	5×10^{-4}	5	0.37	89.56
	10^{-2}	10^{-4}	4.5	0.36	89.66
ResNet-18	10^{-1}	–	4.2	**0.21**	**94.17**
	10^{-2}	–	3.4	0.22	93.69
	10^{-3}	–	6	0.26	92.50
	10^{-4}	–	23.6	0.30	91.02
	10^{-1}	10^{-3}	3.8	0.20	94.25

	10^{-1}	5×10^{-4}	3.2	**0.20**	**94.30**
	10^{-1}	10^{-4}	3.8	0.21	94.01
VGG-11	10^{-1}	–	None	None	None
	10^{-2}	–	8	**0.27**	**92.30**
	10^{-3}	–	37.8	0.26	90.34
	10^{-2}	10^{-3}	15.8	0.32	91.32
	10^{-2}	5×10^{-4}	8.6	0.30	91.94
	10^{-2}	10^{-4}	9.4	**0.30**	**92.53**

4.5 SVHN Experiments

We apply the experimental configuration of CIFAR-10 on the SVHN dataset. The only difference is in the architecture of the "small CNN", which is described in Section 4.2.

Baseline macro-CV results are presented in Table 3. Micro-CV results are presented in Table 4. The summary of results is similar with that of CIFAR-10. In terms of test accuracy, macro-CV outperforms micro-CV, however, when time cost and test accuracy are considered together, micro-CV achieves a high relative test-accuracy with lower time cost. For the small CNN, micro-CV achieves 89.55% test accuracy, which is 99% of macro-CV's test accuracy (90.16%) in only 21% of the time spent by macro-CV (87min 47s vs. 36min 30s). Similarly, for ResNet-18, micro-CV achieves 98% relative accuracy with 45% relative time cost; for VGG-11, these numbers are 93% and 61%.

Table 4 Comparison of macro-CV and micro-CV results on SVHN. 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**
Small CNN	Macro-CV baseline	4.8	0.36	**90.16**	(440K, 220K)	36min 30s
	Micro-CV $\lambda = 1$	27.8	0.37	89.55	(137K, 109K)	14min 25s
	Micro-CV $\lambda = 0.9$	33.4	0.62	89.15	(260K, 122K)	17min 37s
	Micro-CV $\lambda = 0.7$	6	0.55	83.30	(126K, 60K)	8min 35s
	Micro-CV $\lambda = 0.5$	3.6	1.20	58.85	**(115K, 54K)**	**7min 47s**
ResNet-18	Macro-CV baseline	3.2	0.20	**94.30**	(388K, 194K)	2h 47 min
	Micro-CV $\lambda = 1$	80.5	0.40	88.53	(286K, 229K)	2h 2 min
	Micro-CV $\lambda = 0.9$	92.6	0.31	91.45	(496K, 229K)	2h 53 min
	Micro-CV $\lambda = 0.7$	66.25	0.38	93.78	(420K, 197K)	2h 29min
	Micro-CV $\lambda = 0.5$	23.4	0.27	92.73	**(211K, 99K)**	**1h 15min**
VGG-11	Macro-CV baseline	9.8	0.30	**92.53**	(436K, 218K)	1h 37min
	Micro-CV $\lambda = 1$	100	0.45	86.25	(286K, 229K)	59min 55s
	Micro-CV $\lambda = 0.9$	13.7	1.46	48.70	(163K, 77K)	25min 54s
	Micro-CV $\lambda = 0.7$	3.4	2.05	26.30	(114K, 54K)	17min 59s
	Micro-CV $\lambda = 0.5$	2	2.23	19.60	**(107K, 50K)**	**16min 54s**

Table 5 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.

LR	**LR Decay**	**Epochs**	**Test Loss**	**Test Acc.**
10^{-1}	–	26.6	1.47	50.12
10^{-2}	–	12.8	1.65	48.64
10^{-3}	–	16.4	1.68	44.81
10^{-4}	–	77	1.71	42.61

10^{-1}	10^{-2}	139.5	1.44	48.64
10^{-1}	10^{-3}	45.6	1.46	49.53
10^{-1}	5×10^{-4}	65.4	1.54	45.86
10^{-1}	10^{-4}	24.4	**1.37**	**51.02**

4.6 Adience Experiments

Compared to CIFAR-10 and SHVN, the Adience dataset contains higher resolution images. Therefore, on this dataset, we use a larger network, namely, ResNet-50.

Train and test experiment configuration is the same as that of CIFAR-10. One difference here is that experiments are repeated only 3 times, instead of 5. Another difference is in the selection of the validation set for the macro-CV experiments. Since train, test and validation sets are provided in the dataset, we used the provided validation set in all experiments instead of random validation set for each experiment. Baseline macro-CV results are presented in Table 5. And, the micro-CV results are presented in Table 6.

Table 6 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**
Macro-CV baseline	24.4	1.37	51.02	(522K, 261K)	26h 16min
Micro-CV $\lambda = 1$	75.6	1.80	39.11	(176K, 141K)	**3h 38min**
Micro-CV $\lambda = 0.9$	96.7	1.63	46.75	(314K, 148K)	4h 13min
Micro-CV $\lambda = 0.7$	81.6	2.14	48.20	(314K, 148K)	4h 13min
Micro-CV $\lambda = 0.5$	72.6	2.28	**52.50**	**(291K, 137K)**	3h 54min

5. Conclusion

In this paper, we propose a minibatch-level (i.e. micro) cross-validation algorithm that can dynamically and adaptively choose the learning rate at each training epoch. As an alternative to the dataset-level, macro approach to cross-validation, in micro-CV, the randomly sampled minibatch is randomly split into training and validation halfbatches. The gradient vector computed on the training half-batch is re-used to evaluate several different learning rates (i.e. step sizes) on the validation half-batch. Using stochastic gradient descent with momentum as the optimizer, experiments on three different datasets with three different neural network architectures show the potential of our micro-CV training algorithm.

On CIFAR-10 and SVHN, micro-CV achieves a slightly lower test accuracy and macro-CV, however, it spends a much lower computational budget and time. For example, on CIFAR-10 using a ResNet-18 network, while macro-CV achives 81.50% test accuracy in 2 hours 2 minutes, micro-CV achieves 81.03% in 1 hour 25 minutes. On SVHN, with the same architecture, macro-CV yields 94.30% accuracy in 2 hours 47 minutes, micro-CV yields 92.73% in 1 hour 15 minutes. Finally, on the relatively much larger and more realistic Adience dataset, our micro-CV algorithm outperforms macro-CV with a drastic reduction in total training time; macro-CV: 51.02% test accuracy in 26 hours 16 minutes, micro-CV : 52.50% test accuracy in 3 hours 54 minutes. These results show the promising potential of our micro-CV algorithm. Futher research is required to comprehensively compare micro-CV to adaptive learning rate methods such as Adam, AdaGrad and Rmsprob.

Acknowledgments

The numerical calculations reported in this paper were fully performed at TUBITAK ULAKBIM, High Performance and Grid Computing Center (TRUBA resources).

References

[1] K. Anand, Z. Wang, M. Loog, and J. van Gemert, "Black magic in deep learning: How human skill impacts network training," in *British Machine Vision Conference*, 2020.

[2] R. Schwartz, J. Dodge, N. A. Smith, and O. Etzioni, "Green ai," *Communications of the ACM*, vol. 63, p. 54–63, Nov. 2020.

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

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

[5] "Neural networks (maybe) evolved to make adam the best optimizer – parameter-free learning and optimization algorithms." https://parameterfree.com/2020/12/06/ neural-network-maybe-evolved-to-make-adam-the-best-optimizer/. (Accessed on 03/01/2021).

[6] 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.

[7] J. Zhang, I. Mitliagkas, and C. Re, "YellowFin and the Art of Momentum Tuning,"′ *CoRR*, vol. abs/1706.0, 2017.

[8] D. Kabakc¸ı, "Automated learning rate search using batch-level cross-validation," Master's thesis, Middle East Technical University, Ankara, Turkey, July 2019. https://open.metu.edu.tr/handle/11511/43629.

[9] 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.

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

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

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

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

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

[15] 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.

[16] N. S. Keskar and R. Socher, "Improving generalization performance by switching from Adam to SGD," *arXiv preprint arXiv:1712.07628*, 2017.

[17] M. Zaheer, S. Reddi, D. Sachan, S. Kale, and S. Kumar, "Adaptive Methods for Nonconvex Optimization," in *Advances in Neural Information Processing Systems 31* (S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, eds.), pp. 9793–9803, Curran Associates, Inc., 2018.

[18] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[19] 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.

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

[21] Y. Huang, Y. Cheng, D. Chen, H. Lee, J. Ngiam, Q. Le, and Z. Chen, "Gpipe: Efficient training of giant neural networks using pipeline parallelism.," in *Advances in Neural Information Processing Systems (NIPS)*, 2019.

[22] S. Jenni and P. Favaro, "Deep bilevel learning," in *Proceedings of the European conference on computer vision (ECCV)*, pp. 618–633, 2018.

[23] 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.

[24] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[25] A. Krizhevsky, G. Hinton, *et al.*, "Learning Multiple Layers of Features from Tiny Images," tech. rep., Department of Computer Science, University of Toronto, 2009.

[26] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y Ng, "Reading Digits in Natural Images with Unsupervised Feature Learning," in *Advances in Neural Information Processing Systems (NIPS)*, 2011.

[27] 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.

[28] 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.