AUTODROP: TRAINING DEEP LEARNING MODELS WITH AUTOMATIC LEARNING RATE DROP

Anonymous authors
Paper under double-blind review

ABSTRACT

Modern deep learning (DL) architectures are trained using variants of the SGD algorithm that is run with a *manually* defined learning rate schedule, i.e., the learning rate is dropped at the pre-defined epochs, typically when the training loss is expected to saturate. In this paper we develop an algorithm that realizes the learning rate drop *automatically*. The proposed method, that we refer to as AutoDrop, is motivated by the observation that the angular velocity of the model parameters, i.e., the velocity of the changes of the convergence direction, for a fixed learning rate initially increases rapidly and then progresses towards soft saturation. At saturation the optimizer slows down thus the angular velocity saturation is a good indicator for dropping the learning rate. After the drop, the angular velocity "resets" and follows the previously described pattern - it increases again until saturation. We show that our method improves over SOTA training approaches: it accelerates the training of DL models and leads to a better generalization. We also show that our method does not require any extra hyperparameter tuning. AutoDrop is furthermore extremely simple to implement and computationally cheap. Finally, we develop a theoretical framework for analyzing our algorithm and provide convergence guarantees.

1 INTRODUCTION

As data sets grow in size and complexity, it is becoming more difficult to pull useful features from them using hand-crafted feature extractors. For this reason, DL frameworks (Goodfellow et al., 2016) are now widely popular. DL frameworks process input data using multi-layer networks and automatically find high-quality representation of complex data useful for a particular learning task. Today DL approaches are generally recognized as superior to all alternatives for image (Krizhevsky et al., 2012; He et al., 2016), speech (Abdel-Hamid et al., 2012), and video (Karpathy et al., 2014) recognition, image segmentation (Chen et al., 2016), and natural language processing (Weston et al., 2014). Furthermore, DL is the leading artificial intelligence technology in major tech companies such as Facebook, Google, Microsoft, and IBM, as well as in countless start-ups, where it is used for a plethora of learning problems including content filtering, photo collection management, topic classification, search/ad ranking, video search and indexing, and copyrighted material detection.

Setting the values and schedules of the hyperparameters for training DL models is computationally expensive and time consuming, e.g., a deep model with around ten billion parameters requires roughly 500 GPUs to be trained in around two weeks (Shoeybi et al., 2019). Among all hyperparameters used when training DL models, the learning rate schedule is one of the most important (Jin et al., 2020). For most SOTA DL architectures, the learning rate is dropped several times during training at epochs chosen by the user. With growing sizes of modern architectures however, performing any manual tuning of the hyperparameters will eventually become prohibitive. More efficient techniques that allow automatic and online setting of hyperparameters translate to substantial savings of resources, time, and money (today the cost of training a single state-of-the-art DL model reaches up to hundreds of thousands of dollars (Peng, 2019)).

This paper addresses a challenge of developing an automatic method for adjusting the learning rate that works in an online fashion during network training and does not introduce any extra hyper-parameters to tune. The basis for our approach is rooted in the observation that the angular velocity of the model parameters, defined below, is an excellent indicator of the dynamics of the convergence

of an optimizer and can be easily used to guide the learning rate drop during network training. The resulting algorithm that we obtain is extremely simple, can be used on the top of any DL optimizer (SGD (Bottou, 1998), momentum SGD (Polyak, 1964), ADAM (Kingma & Ba, 2015), etc.), and enjoys an elegant theoretical framework. We empirically demonstrate that our method accelerates the training of DL models and leads to better generalization compared to SOTA techniques.

Definition 1. *Define the angular velocity of model parameters as:*

$$\omega_i = \frac{\angle(s_i, s_{i-1})}{1 \;\; epoch}, \quad where \quad s_i = x_{i+1} - x_i \tag{1}$$

and x_i is the parameter vector in the end of the i^{th} epoch. The operator $\angle(\cdot, \cdot)$ calculates the angle between two vectors and is defined as:

$$\angle(s_i, s_{i-1}) = \frac{180^\circ}{\pi} \cdot \arccos\left(\frac{s_i^T s_{i-1}}{||s_i|| ||s_{i-1}|| + \epsilon}\right), \tag{2}$$

where ϵ is a small positive number preventing the division by zero [1].

This paper is organized as follows: Section 2 discusses the related work, Section 3 builds an intuition for understanding our algorithm based on simple examples, Section 4 shows our algorithm, Section 5 captures the theoretical convergence guarantees, Section 6 presents experimental results, and Section 7 concludes the paper. All proofs and experimental details are deferred to the Supplement.

2 Related Work

In this section, we summarize different types of learning rate adaptation methods and divide them into four major categories. *Scheduling-based methods* rely on a carefully designed learning rate schedules that are tailored to the non-convex nature of the deep learning optimization. More specifically, it was proposed in (Smith, 2017) to use cyclical learning rate pattern to train DL models and apply a triangular learning rate policy in each cycle (i.e., first increase and then decrease the learning rate linearly in the cycle) to potentially allow more rapid traversal of saddle point plateaus. This idea was further extended to the super-convergence policy (Smith & Topin, 2017) where there is only one triangular cycle for the whole training process. This concept was also applied to other hyperparameters, e.g.:, momentum coefficient (Smith, 2018). Cyclical learning rates were also used in (Loshchilov & Hutter, 2017), where the authors combine them with restart techniques when training deep neural networks. The authors decrease the learning rate from a maximum value to a minimum value using a cosine annealing scheme and then periodically restart the process. All these methods define the learning rate policy manually, thus they constitute deterministic scheduling methods. As opposed to these techniques, (Jin et al., 2020) proposes an automatic learning rate scheduling method. The authors use Gaussian process as a surrogate model to establish the connection between the learning rate and the expected validation loss. The method updates a posterior distribution of the validation loss repeatedly and search for the best learning rate with respect to the posterior on the fly. This method requires a careful design of an acquisition function and a forecasting model in order to obtain an accurate prediction of the posterior of the validation loss.

Another group of techniques are *hypergradient-based methods* (Donini et al., 2020; Yang et al., 2019; Baydin et al., 2018; Franceschi et al., 2017) that optimize both the model parameters and the learning rate simultaneously. The authors of these methods typically introduce a hypergradient that is defined as a gradient of the validation error with respect to the learning rate schedule. The learning rate is optimized online via gradient descent. This technique however is quite sensitive to the choice of the hyperparameters and is usually unable to reach state-of-the-art performance (Jin et al., 2020).

Hyperparameter optimization methods aim to automatically find a good set of hyperparameters offline. They either build explicit regression models to describe the dependence of target algorithm performance on hyperparameter settings (Hutter et al., 2011), or optimize hyperparameters by performing random search along with using greedy sequential methods based on the expected improvement criterion (Bergstra et al., 2011), or use bandit-based approach for hyperparameter selection (Li et al., 2018). These technique can be combined with Bayesian optimization (Falkner et al., 2018; Zela et al., 2018). Recently, several parallel methods were proposed for hyperparameter tuning (Jaderberg et al., 2017; Li et al., 2019; Parker-Holder et al., 2020; Li et al., 2020) as well. The hyperparameter optimization methods are computationally expensive in practice.

[1] ϵ is omitted in the theoretical derivations.

Finally, popular *adaptive learning rate optimizers* adjust the learning rate for each parameter individually based on gradient information from past iterations. AdaGrad (Duchi et al., 2011) proposes to update each parameter using different learning rate which is proportional to the inverse of the past accumulated squared gradients of the parameter. Thus the parameters associated with larger accumulated squared gradients have smaller step size. This method is enabling the model to learn infrequently occurring features, as these features might be highly informative and discriminative. The major weakness of AdaGrad is that the learning rates continually decrease during the training and eventually become too small for the model to learn. Later on, RMSprop (Tieleman et al., 2012) and Adadelta (Zeiler, 2012) were proposed to resolve the issue of diminishing learning rate in AdaGrad. Instead of directly summing up the past squared gradients, both methods maintain an exponential average of the squared gradients which is used to scale the learning rate of each parameter. The exponential average of the squared gradients could be considered as an approximation to the second moment of the gradients. One step further, ADAM (Kingma & Ba, 2015) estimates both first and second moments of the gradients and use them together to update the parameters.

3 Motivating Example

In this section we analyze the properties of the angular velocity for a noisy quadratic model. While simple, this model is used as a proxy for analyzing neural network optimization (Schaul et al., 2013; Martens & Grosse, 2015; Zhang et al., 2019b).

Definition 2 (Noisy Quadratic Model)**.** *We use the same model as in (Zhang et al., 2019b). The model is represented by the following loss function*

$$L(x) = \frac{1}{2}(x-c)^T A(x-c), \tag{3}$$

where $c \sim N(x^*, \Sigma)$ *and both* A *and* Σ *are diagonal. Without loss of generality, we assume* $x^* = 0$*.*

The update formula for the gradient descent at the step $t+1$ is given as

$$x_{t+1} = x_t - \alpha \nabla L(x_t) = x_t - \alpha A(x_t - c_t), \quad c_t \sim N(0, \Sigma), \tag{4}$$

where α is the learning rate.

We optimize noisy quadratic model with $x \in \mathbb{R}^{200}$ and $A = diag(\frac{1}{10}, \frac{2}{10}, ..., \frac{200}{10})$ using Gradient Descent (GD), where in each experiment $\alpha = [0.06, 0.03, 0.01, 0, 001]$.

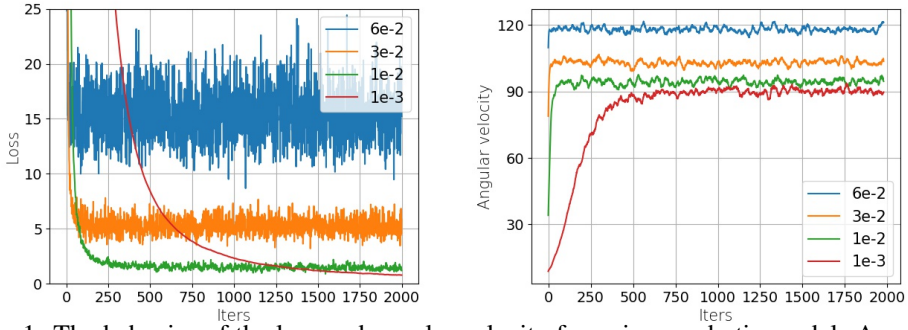

Figure 1: The behavior of the loss and angular velocity for noisy quadratic model. An optimizer is run with different settings of the learning rate $\alpha = [0.06, 0.03, 0.01, 0, 001]$. Angular velocity is averaged over 20 iterations.

The experiments captured in Figure 1 reveal the following properties:

(P1) **Angular velocity saturation:** the angular velocity curves[2] have the tendency to saturate as the training proceeds, and furthermore when the angular velocity enters the saturation phase, the optimizer slows down its convergence,

(P2) **Angular velocity saturation levels:** i) if the learning rate is large enough such that the algorithm cannot converge to the optimum, the angular velocity saturates at a level larger than 90 degrees and smaller than 120 degrees; ii) as the learning rate decreases, and the algorithm

[2]For the noisy quadratic model, the angular velocity (given in Definition 1) is computed with respect to one iteration, rather than an epoch, as for this model there is no notion of the epoch.

systematically converges closer to the optimum, the angular velocity saturates at progressively lower levels; iii) smaller learning rate leads to a slower saturation of the angular velocity; iv) when the learning rate is low enough such that the algorithm can converge to the optimum, the angular velocity saturates at 90 degrees.

These empirical properties can be theoretically justified as shown in the next theorem.

Theorem 1. *Let the i-th diagonal terms of matrices A and Σ in the noisy quadratic model be given as a_i and σ_i, respectively. Then, the expected inner product $< s_t, s_{t+1} >$ converges to*

$$I^* = \lim_{t\to\infty} \mathbb{E}[< s_t, s_{t+1} >] = -\alpha^3 \sum_{i=1}^{n} \frac{a_i^3 \sigma_i^2}{2 - \alpha a_i}. \tag{5}$$

Moreover, the cosine value of an angle between two consecutive steps $\cos \angle(s_t, s_{t+1})$ satisfies

$$C^* = \lim_{t\to\infty} \mathbb{E}[cos(\angle(s_t, s_{t+1}))] \approx -\frac{\alpha}{2} \frac{\sum_{i=1}^{n} \frac{a_i^3 \sigma_i^2}{2-\alpha a_i}}{\sum_{i=1}^{n} \frac{a_i^2 \sigma_i^2}{2-\alpha a_i}} \geq -\frac{\alpha \max_i a_i}{2} \tag{6}$$

$C^* \in [-\frac{1}{2}, 0]$ and thus $\angle(s_t, s_{t+1})$ is between 90 to 120 degrees.

Theorem 1 implies that as training proceeds, the angular velocity eventually saturates as stated in property P1. Theorem 1 furthermore shows that decreasing the learning rate causes the angle between s_t and s_{t+1} to converge to a smaller value. Also, from Theorem 1, $I^* = \lim_{t\to\infty} \mathbb{E}[< s_t, s_{t+1} >] = -\sum_{i=1}^{n} (\alpha a_i)^3 \sigma_i^2 \left[\frac{1}{2-\alpha a_i}\right]$. When $\alpha a_i (i = 1, .., n)$ is small enough, I^* can be treated as 0 which implies that s_t is orthogonal to s_{t+1}. In other words, the angle between s_t and s_{t+1} converges to 90 degrees for small enough learning rate. Otherwise, for larger learning rates, this angle saturates above 90 degrees. Furthermore, the limit of cosine angle C^* is approximately larger than $-\frac{1}{2}$, thus the saturation level of angular velocity should be below 120 degrees. This together supports property P2 (in particular this supports points i,ii, and iv; point iii remains an empirical observation).

We next empirically verified whether these observations carry over to non-convex DL setting on a simple experiment reported in Figure 2. Clearly, property P1 holds, whereas property P2 is satisfied partially. In particular conclusion iii is broken as the angular velocity may not reach 90 degrees. Also, in a DL setting one can observe that for lower learning rates the angular velocity curves become more noisy at saturation, which was not the case for a noisy quadratic model.

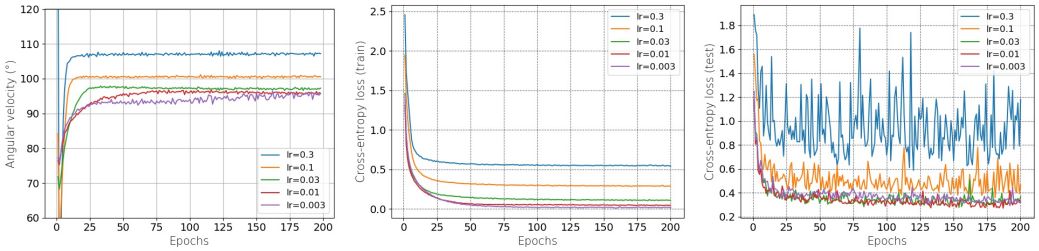

Figure 2: The behavior of the loss and angular velocity for an exemplary DL problem (training ResNet-18 on CIFAR-10). An optimizer is run with different settings of the learning rate $\alpha = [0.3, 0.1, 0.03, 0.01, 0.003]$. Angular velocity is calculated over a single epoch.

Property P1 is a key observation underlying our algorithm. An important conclusion from this observation is that the saturation of the angular velocity can potentially guide the drop of the learning rate of the optimization algorithm. In other words, given the lower-bound on the learning rate, each time the angular velocity saturates, the learning algorithm should decrease the learning rate. Tracking the saturation of the angular velocity is more plausible than tracking the saturation of the loss function since, as can be clearly seen in Figure 1, angular velocity curves follow much harder saturation pattern. Also, the loss function does not necessary need to have a bounded range, as opposed to the angular velocity. We found that property P1 is sufficient to design an optimization algorithm for training DL models. The algorithm is described in Section 4. Property P2 is crucial for the theoretical analysis provided in Section 5.

Following the above intuition, we implement a simple algorithm for optimizing the noisy quadratic model. The algorithm drops the learning rate by a factor of 2 when the angular velocity saturates (i.e.:, the change of the angular velocity averaged across 20 iterations is smaller than 0.01 degree between

2 consecutive iterations). The initial learning rate was set to 0.06 and the minimal one was set to 0.001. Figure 3 captures the results. It shows that the algorithm that is using the angular velocity to guide the drop of the learning rate indeed converges to the optimum.

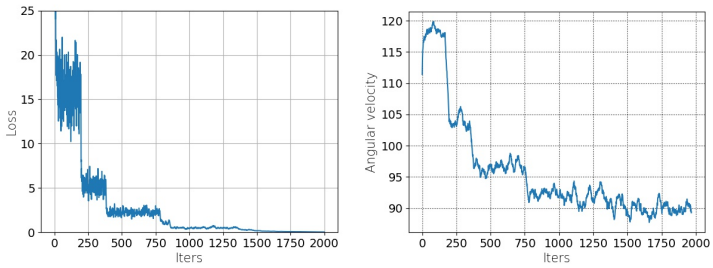

Figure 3: The behavior of the loss and angular velocity for the noisy quadratic model. An optimizer is using an automatic drop of the learning rate guided by the saturation of the angular velocity. Angular velocity is averaged over 20 iterations.

The aforementioned simple algorithm led us to derive the method for optimizing DL models using automatic learning rate drop that we refer to as AutoDrop. The obtained method is a straightforward extension of the above algorithm and is described in the next section. The extension accommodates the fundamental difference that we observed between noisy quadratic model and the DL model: the fact that in the case of DL models, lower learning rates lead to a larger noise of the angular velocity at saturation.

Algorithm 1 AutoDrop

Require:
α_0 and $\underline{\alpha}$: initial learning rate of the optimizer and its lower bound
θ_0 and $\overline{\theta}$: initial threshold for the change in the angular velocity and its upper bound
ρ: learning rate drop factor
x_0 : initial model parameter vector
n_d: learning rate drop delay in number of epochs

$x \leftarrow x_0$, $\alpha \leftarrow \alpha_0, \theta \leftarrow \theta_0$, $s_0 \leftarrow 0$, $t \leftarrow 0$
$drop\alpha \leftarrow false$
while not converged **do**
  $t \leftarrow t + 1$, $y \leftarrow x$
  Train the model for one epoch with learning rate α and update x accordingly
  $s_t \leftarrow x - y$; $\omega_t \leftarrow \angle(s_t, s_{t-1})$

  //Check the condition for dropping α
  if $t > 2$ and $|\omega_t - \omega_{t-1}| < \theta$ **then**
    $z \leftarrow 0$, $k \leftarrow 0$, $drop\alpha \leftarrow true$
  end if

  if $drop\alpha$ **then**
    $k \leftarrow k + 1$
    $z \leftarrow z + k \cdot x$ //Accumulate the scaled values of parameters
    if $k \geq n_d$ **then**
      $x \leftarrow \frac{2z}{(n_d+1)n_d}$ //Compute the exponential average of model parameters
      $\alpha \leftarrow \max\{\underline{\alpha}, \rho \times \alpha\}$, $\theta \leftarrow \min\{\overline{\theta}, \frac{1}{\rho} \times \theta\}$ //Drop α and adjust θ
      $drop\alpha \leftarrow false$
    end if
  end if
end while

//Recommended setting of hyperparameters:
$\underline{\alpha} = 0.0001$, $\theta_0 = 0.01°$, $\overline{\theta} = 1°$, and $n_d = 20$
ρ, α_0 - the same as in SOTA

4 ALGORITHM

The algorithm for training deep learning models with automatic learning rate drop is captured in Algorithm 1. The algorithm admits on its input the initial learning rate α_0, the value of the smallest permissible learning rate $\underline{\alpha}$, initial threshold for the change in the angular velocity θ_0 that will determine the first drop of the learning rate, the value of the largest permissible threshold for the change in the angular velocity $\overline{\theta}$, learning rate drop factor ρ ($\rho \in (0, 1)$; each time the learning rate is dropped, it is multiplied by ρ), initial model parameter vector x_0, and the learning rate drop delay n_d (this hyper-parameter will be explained in the next paragraph).

The algorithm triggers the procedure for dropping the learning rate each time the angular velocity changes by less than the threshold θ between two consecutive epochs (θ is initialized with θ_0). Before the learning rate is dropped (i.e., multiplied by ρ), the optimizer continues operating with the current learning rate for another n_d epochs during which it calculates the exponential average of model parameters (parameter averaging is commonly done by practitioners and was proposed by (Polyak & Juditsky, 1992)). This is done to stabilize the learning process. Finally, after each learning rate drop, the threshold for the change in the angular velocity is increased (i.e., divided by ρ). This is necessary as the angular velocity becomes more noisy for the lower learning rates.

AutoDrop algorithm can be thought of as a meta-scheme that can be put on the top of any optimization method for training deep learning models. Thus one can use any optimizer to update model parameters. In practice we recommend using the following setting of the hyperparameters for our algorithm: $\underline{\alpha} = 0.0001$, $\theta_0 = 0.01°$, $\overline{\theta} = 1°$, $n_d = 20$, and ρ set in the same way as in SOTA. As will be shown in the experimental section this set of parameters guarantees good performance for a wide range of model architectures and data sets.

5 THEORY

This section theoretically shows that decreasing the learning rate when the angular velocity saturates guarantees the sub-linear convergence rate of SGD and momentum SGD methods.

5.1 UNIFIED CONVERGENCE ANALYSIS FOR SGD AND MOMENTUM SGD WITH DISCRETE LEARNING RATE DROP

Firstly, we present a unified theoretical framework that covers the update rule of both SGD and momentum SGD. We refer to these update rules jointly as Unified Momentum (UM) method. This framework was proposed in (Yang et al., 2016).

$$\text{UM}: \quad \begin{cases} y_{t+1} = x_t - \alpha_t \mathcal{G}(x_t; \xi_t) \\ y^s_{t+1} = x_t - s\alpha_t \mathcal{G}(x_t; \xi_t) \\ x_{t+1} = y_{t+1} + \beta(y^s_{t+1} - y^s_t) \end{cases} \tag{7}$$

where t is the iteration index, β is the momentum parameter, α_t is the learning rate at time t, x_t is the parameter vector at time t, and $\mathcal{G}(x_t; \xi_t)$ is the gradient of the loss function at time t computed for a data mini-batch ξ_t. s is the factor that controls the type of optimization method in the following way:

- $s = 0$ Heavy-Ball (HB) method:

$$\text{HB:} \quad x_{t+1} = x_t - \alpha_t \mathcal{G}(x_t; \xi_t) + \beta(x_t - x_{t-1})$$

- $s = 1$ Nestrov (NAG) method:

$$\text{NAG:} \quad \begin{cases} y_{t+1} = x_t - \alpha_t \mathcal{G}(x_t; \xi_t) \\ x_{t+1} = y_{t+1} + \beta(y_{t+1} - y_t) \end{cases}$$

- $s = 1/(1 - \beta)$ Gradient Descent (GD) method:

$$\text{GD:} \quad x_{t+1} = x_t - \alpha_t/(1 - \beta)\mathcal{G}(x_t; \xi_t).$$

The state-of-the-art convergence analysis for common machine learning optimizers only supports constant learning rate (Le Roux et al., 2012; Yang et al., 2016; Schmidt et al., 2017; Ramezani-Kebrya et al., 2018; Zhang et al., 2019a) or continuous learning rate drop schemes (Wu et al., 2018; 2019; Gower et al., 2019). However, the learning rate is dropped in a discrete fashion in many practical cases, especially in DL. Theorem 2 provides a theoretical convergence guarantee for optimization algorithms that use discrete learning rate drop. The theorem requires some mild (easy to satisfy

in practice and thus realistic) constraints on the drop gap (k_i), i.e.:, the frequency of dropping the learning rate. Theorem 2 accommodates learning settings relying on discrete learning rate drops and thus is well-aligned with approaches used by practitioners. Moreover, in the next section we extend this theorem to our AutoDrop approach.

Theorem 2. *Suppose $f(x)$ is a convex function, $\mathbb{E}\left[\|\mathcal{G}(x;\xi) - \mathbb{E}[\mathcal{G}(x;\xi)]\|\right] \leq \delta^2$ and $\|\partial f(x)\| \leq G$ for any x and some non-negative G. Given a sequence of decreasing learning rates $\{\hat{\alpha}_i\}_{i=-1}^{n-1} \subset (0,1)$ and a sequence of integers $\{k_i\}_{i=0}^{n-1} \subset \mathbb{N}$ $(n \gg 1)$, there exits constants κ_1, κ_2 such that*

$$\hat{\alpha}_i \leq (i+2)^{-\frac{2}{3}}, \quad k_i\hat{\alpha}_i \geq \kappa_1(i+2)^{-\frac{1}{3}}, \quad k_i\hat{\alpha}_i\hat{\alpha}_{i-1} \leq \kappa_2(i+1)^{-1}, \quad \forall i = 0, 1, ..., n-1. \tag{8}$$

Define a partition $\Pi : 0 = t_0 < t_1 < ... < t_n = T (T = \sum_{i=0}^{n-1} k_i)$ based on the integer sequence $\{k_i\}_{i=0}^{n-1}$ such that the gap between t_i and t_{i+1} is k_i ($k_i = t_{i+1} - t_i$). Run UM update defined in Equation 7 for T iterations by setting the learning rate α_t based on a sequence $\{\hat{\alpha}_i\}_{i=-1}^{n-1}$ as

$$\alpha_t = \hat{\alpha}_i, \quad \text{where } t_i \leq t < t_{i+1}. \tag{9}$$

Then the following holds:

$$\min_{t=0,\ldots,T-1}\{\mathbb{E}[f(x_t) - f(x^*)]\} \leq \frac{2\beta(f(x_0) - f(x^*))[(n+1)^{\frac{1}{3}} - 2^{\frac{1}{3}}]}{2\kappa_1(1-\beta)[(n+1)^{\frac{2}{3}} - 2^{\frac{2}{3}}]} + \frac{(1-\beta)\|x_0 - x^*\|^2}{3\kappa_1[(n+1)^{\frac{2}{3}} - 2^{\frac{2}{3}}]} + \frac{(2s\beta+1)(G^2+\delta^2)\kappa_2 \log n}{3(1-\beta)\kappa_1[(n+1)^{\frac{2}{3}} - 2^{\frac{2}{3}}]}. \tag{10}$$

5.2 Convergence Analysis of AutoDrop

For a fixed learning rate α, we introduce a simplified mathematical model of the behavior of the angular velocity as a function of iterations. The model is defined below (and depicted in Figure 4):

$$v_\alpha(t) = \frac{\pi}{2}(1+\epsilon\alpha)\left(1 - \frac{1}{\gamma\alpha t}\right), \tag{11}$$

where t is the number of iterations, ϵ and γ are two constants that control the asymptote and curvature of the velocity.

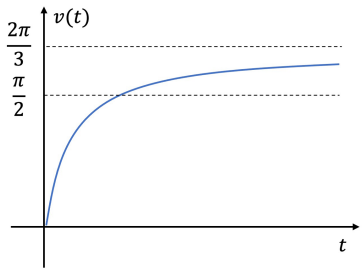

Figure 4: Angular velocity model for a fixed learning rate α.

Algorithm 2 AutoDrop (approximate)

Inputs: x_0: initial weight
Hyperparameters: $\{\hat{\alpha}_i\}$: set of learning rates, $v_\alpha(t)$: ang. vel. model, τ: threshold for the derivative of ang. vel.
Initialize $i = 0$, $t_0 = 0$, $t = 0$
while $i < n$ **do**
  Update x_t via (7) with learning rate $\alpha_t = \hat{\alpha}_i$.
  if $v'_{\hat{\alpha}_i}(t - t_i) \leq \tau$ **then**
    $i = i + 1; t_i = t$
  end if
  $t = t + 1, T = t$
end while
return $\{x_t\}_{t=0}^{T-1}$ (T: # iterations)

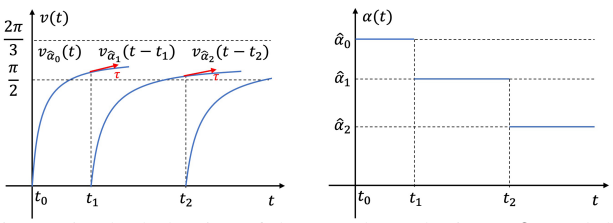

Figure 5: The behavior of the angular velocity (**left**) and the learning rate (**right**) for Algorithm 2.

$v_\alpha(t)$ saturates in $\frac{\pi}{2}[1+\epsilon\alpha]$ when t goes to infinity. Note that the given model complies with the property P2 empirically observed and described in Section 3: i) if the learning rate is large enough, the angular velocity saturates at a level larger than $\pi/2$ and smaller than $2\pi/3$; ii) as the learning rate decreases, the angular velocity saturates at progressively lower levels; iii) smaller learning rate leads to a slower saturation of angular velocity; iv) when the learning rate is low enough the angular velocity saturates at $\pi/2$. Lets assume an upper-bound α_{max} for the learning rate. Since the limit of the angular velocity should be between $\pi/2$ and $2\pi/3$, the range of factor ϵ is set to be $(0, \frac{1}{3\alpha_{max}})$.

For the the purpose of the theoretical analysis, we drop the learning rate every time the derivative of the angular velocity decreases to a threshold τ (Algorithm 2) instead of detecting whether the change of the angular velocity is small enough (Algorithm 1). Intuitively, when the derivative of the angular velocity is close to zero, we would expect the angular velocity to saturate. We are going to analyze the convergence of Algorithm 2, which is an approximate version of Algorithm 1. The behavior of the angular velocity and the learning rate for Algorithm 2 is depicted in Figure 5.

Theorem 3. *Suppose $f(x)$ is a convex function, $\mathbb{E}\left[\|\mathcal{G}(x;\xi) - \mathbb{E}[\mathcal{G}(x;\xi)]\|\right] \leq \delta^2$ and $\|\partial f(x)\| \leq G$ for any x and some non-negative G. Given the sequence of the learning rates $\{\hat{\alpha}_i\}_{i=-1}^{n-1}$ such that $\hat{\alpha}_i = (i+1)^{-\frac{2}{3}}$, parameters $\epsilon \in (0, \frac{1}{3\hat{\alpha}_0})$ and γ defining the angular velocity model $v_\alpha(t)$ (Equation 11), and the threshold τ for the derivative of the angular velocity, the sequence of weights $\{x_t\}_{t=0}^{T-1}$ generated by Algorithm 2 satisfies*

$$\min_{t=0,\dots,T-1}\{\mathbb{E}[f(x_t)-f(x^*)]\} \leq \frac{2\beta(f(x_0)-f(x^*))[(\frac{5T}{3\kappa_1})^{\frac{3}{5}}+2)^{\frac{1}{3}}-2^{\frac{1}{3}}]}{2\kappa_1(1-\beta)[(\frac{5T}{3\kappa_2})^{\frac{2}{5}}-2^{\frac{2}{3}}]} + \frac{(1-\beta)\|x_0-x^*\|^2}{3\kappa_1[(\frac{5T}{3\kappa_2})^{\frac{2}{5}}-2^{\frac{2}{3}}]}$$
$$+ \frac{(2s\beta+1)(G^2+\delta^2)\kappa_2\log((\frac{5T}{3\kappa_1})^{\frac{3}{5}}+1)}{3(1-\beta)\kappa_1[(\frac{5T}{3\kappa_2})^{\frac{2}{5}}-2^{\frac{2}{3}}]} \tag{12}$$
$$= O\left(T^{-\frac{1}{5}}\right), \tag{13}$$

where $\kappa_1 = \sqrt{\frac{\pi}{2\gamma\tau}}$ and $\kappa_2 = \sqrt{\frac{2\pi}{3\gamma\tau}}$.

Theorem 3 can be obtained by extending Theorem 2 to the setting accommodating the angular velocity model from Equation 11 and guarantees sub-linear convergence rate of Algorithm 2.

6 Experiments

In this section, we compare the performance of our method, AutoDrop, that automatically adjusts the learning rate, with the SOTA optimization approaches for training DL models that instead manually drop the learning rate. The comparison is performed on the popular DL architectures and benchmark data sets. Our method was run with three different settings of the learning rate drop factor ρ, whereas the remaining hyper-parameters were set as recommended in Section 4. The baselines that we compare with are SOTA approaches taken from the referenced papers that rely on different variants of SGD. Finally, the codes of our method will be publicly released.

Table 1: Test errors of AutoDrop and baselines reported in the literature. For CIFAR-10 and CIFAR-100 we ran each experiment four times with different random seeds. We report the mean and standard deviation of the final test error (at the 200$^{\text{th}}$ epoch). For ImageNet, we ran each experiment once and report the final test error (at the 105$^{\text{th}}$ epoch). † follows the the setup of (Zhang et al., 2019b). ‡ follows the the setup of (Zagoruyko & Komodakis, 2016). * follows the the setup of (He et al., 2016).

Model	Method	Test Error [%]
ResNet-18 CIFAR-10	Baseline† ($\rho = 0.2$)	4.87 ± 0.085
	AutoDrop ($\rho = 0.1$)	5.07 ± 0.465
	AutoDrop ($\rho = 0.2$)	$\mathbf{4.61 \pm 0.173}$
	AutoDrop ($\rho = 0.5$)	$\mathbf{4.71 \pm 0.111}$
WRN-28x10 CIFAR-10	Baseline‡ ($\rho = 0.2$)	3.77 ± 0.05
	AutoDrop ($\rho = 0.1$)	$\mathbf{3.73 \pm 0.26}$
	AutoDrop ($\rho = 0.2$)	$\mathbf{3.73 \pm 0.10}$
	AutoDrop ($\rho = 0.5$)	4.29 ± 1.13
ResNet-34 CIFAR-100	Baseline† ($\rho = 0.2$)	21.91 ± 0.20
	AutoDrop ($\rho = 0.1$)	23.27 ± 0.48
	AutoDrop ($\rho = 0.2$)	$\mathbf{21.82 \pm 0.50}$
	AutoDrop ($\rho = 0.5$)	$\mathbf{21.43 \pm 0.29}$
WRN-40x10 CIFAR-100	Baseline‡ ($\rho = 0.2$)	19.16 ± 0.11
	AutoDrop ($\rho = 0.1$)	$\mathbf{18.25 \pm 0.33}$
	AutoDrop ($\rho = 0.2$)	$\mathbf{18.17 \pm 0.25}$
	AutoDrop ($\rho = 0.5$)	23.15 ± 3.43
ResNet-18 ImageNet	Baseline* ($\rho = 0.1$)	29.93
	AutoDrop ($\rho = 0.1$)	**29.80**

In Table 1 we show the final test errors obtained on CIFAR-10, CIFAR-100, and ImageNet data sets. Our method shows better performance in terms of the final test error compared to the baseline approaches while automatically selecting the epochs for dropping the learning rate. Across all the experiments on CIFAR data sets, AutoDrop run with the learning drop factor $\rho = 0.2$ (the drop factor used by the baselines), was always among the winning AutoDrop strategies. For ImageNet the baseline

recommended using $\rho = 0.1$ and again for this setting AutoDrop performed favorably. Furthermore, in Figure 6 we report an exemplary plot capturing the behavior of the learning rate, train loss, and test error as a function of the number of epochs.

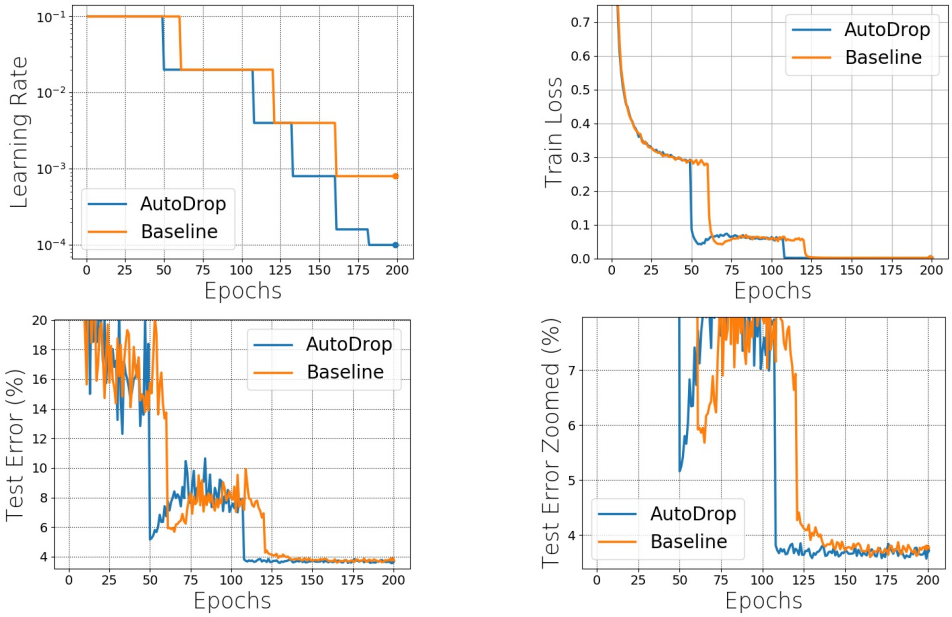

Figure 6: Experimental curves for WRN-28x10 model and CIFAR-10 data set: learning rate, train loss, test error, and zoomed test error.

Table 2: Test error [%] of AutoDrop and the baseline for different initial learning rates. Resnet-18 on CIFAR-10.

Initial LR	Baseline ($\rho = 0.2$)	AutoDrop ($\rho = 0.2$)
0.15	5.04 ± 0.19	$\mathbf{4.90 \pm 0.20}$
0.1	4.80 ± 0.12	$\mathbf{4.62 \pm 0.14}$
0.05	4.87 ± 0.09	$\mathbf{4.61 \pm 0.17}$
0.03	5.07 ± 0.28	$\mathbf{4.73 \pm 0.25}$

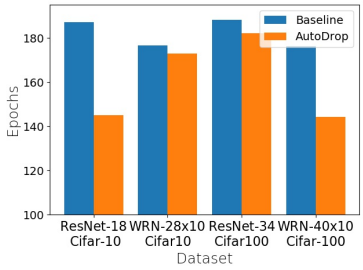

Figure 7: The average number of training epochs needed by an optimizer to achieve the lowest test error. AutoDrop and Baseline use $\rho = 0.2$.

Next, in Table 2 we verify if AutoDrop is more robust to the choice of the initial learning rate than the baseline. We ran an experiment on CIFAR-10 and ResNet-18 and confirmed that indeed across different choices of the initial learning rate, AutoDrop consistently outperforms the baseline.

Finally, Figure 7 is confronting the convergence speed of our method and the baseline by reporting the average number of training epochs needed by AutoDrop and the baseline to achieve the lowest test error. We ran each experiment four times with different random seeds and report the mean value. Clearly, AutoDrop is faster.

7 Conclusions

This paper is motivated by a growing need to develop DL optimization techniques that are more automated in order to increase their scalability and improve the accessibility to DL technology by a wider range of participants. The selection of hyperparameters for training DL models, and especially the learning rate scheduling, is a very hard problem and still remains largely unsolved in the literature. We provide a new algorithm, AutoDrop, for adjusting the learning rate drop during training of DL models that works online and can be run on the top of any DL optimization scheme. It is furthermore a very simple algorithm to implement and use. AutoDrop enjoys favorable empirical performance compared to SOTA training approaches in terms of test error and convergence speed. Finally, our method has a theoretical underpinning that we show, and enjoys sub-linear convergence.

REFERENCES

O. Abdel-Hamid, A.-r. Mohamed, H. Jiang, and G. Penn. Applying convolutional neural networks concepts to hybrid NN-HMM model for speech recognition. In *ICASSP*, 2012.

A. G. Baydin, R. Cornish, D. Martinez Rubio, M. Schmidt, and F. Wood. Online learning rate adaptation with hypergradient descent. In *ICLR*, 2018.

J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl. Algorithms for hyper-parameter optimization. In *NeurIPS*, 2011.

L. Bottou. Online algorithms and stochastic approximations. In *Online Learning and Neural Networks*. Cambridge University Press, 1998.

L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *CoRR*, abs/1606.00915, 2016.

J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR*, 2009.

M. Donini, L. Franceschi, O. Majumder, M. Pontil, and P. Frasconi. Marthe: Scheduling the learning rate via online hypergradients. In *IJCAI*, 2020.

J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(61):2121–2159, 2011.

S. Falkner, A. Klein, and F. Hutter. BOHB: Robust and efficient hyperparameter optimization at scale. In *ICML*, 2018.

L. Franceschi, M. Donini, P. Frasconi, and M. Pontil. Forward and reverse gradient-based hyperparameter optimization. In *ICML*, 2017.

I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. URL http://www.deeplearningbook.org.

R. M. Gower, N. Loizou, X. Qian, A. Sailanbayev, E. Shulgin, and P. Richtárik. Sgd: General analysis and improved rates. In *ICML*, 2019.

K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

F. Hutter, H. H. Hoos, and K. Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *LION*, 2011.

M. Jaderberg, V. Dalibard, S. Osindero, W. M. Czarnecki, J. Donahue, A. Razavi, O. Vinyals, T. Green, I. Dunning, K. Simonyan, C. Fernando, and K. Kavukcuoglu. Population based training of neural networks. *CoRR*, abs/1711.09846, 2017.

Y. Jin, T. Zhou, L. Zhao, Y. Zhu, C. Guo, M. Canini, and A. Krishnamurthy. Autolrs: Automatic learning-rate schedule by bayesian optimization on the fly. In *ICLR*, 2020.

A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and L. Fei-Fei. Large-scale video classification with convolutional neural networks. In *CVPR*, 2014.

D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In *ICLR (Poster)*, 2015.

A. Krizhevsky, V. Nair, and G. Hinton. Cifar-10 and cifar-100 datasets. https://www.cs.toronto.edu/kriz/cifar.html, 2009.

A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012.

N. Le Roux, M. Schmidt, and F. Bach. A stochastic gradient method with an exponential convergence rate for finite training sets. In *NeurIPS*, 2012.

A. Li, O. Spyra, S. Perel, V. Dalibard, M. Jaderberg, C. Gu, D. Budden, T. Harley, and P. Gupta. A generalized framework for population based training. In *ACM SIGKDD*, 2019.

L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *Journal of Machine Learning Research*, 18(185): 1–52, 2018.

L. Li, K. Jamieson, A. Rostamizadeh, E. Gonina, J. Ben-tzur, M. Hardt, B. Recht, and A. Talwalkar. A system for massively parallel hyperparameter tuning. In *Proceedings of Machine Learning and Systems*, 2020.

I. Loshchilov and F. Hutter. SGDR: stochastic gradient descent with warm restarts. In *ICLR*, 2017.

J. Martens and R. Grosse. Optimizing neural networks with kronecker-factored approximate curvature. In *ICML*, 2015.

J. Parker-Holder, V. Nguyen, and S. J. Roberts. Provably efficient online hyperparameter optimization with population-based bandits. In *NeurIPS*, 2020.

T. Peng. The Staggering Cost of Training SOTA AI Models, Technical Report by Medium. https://medium.com/syncedreview/the-staggering-cost-of-training-sota-ai-models-e329e80fa82, 2019.

B. T. Polyak and A. B. Juditsky. Acceleration of stochastic approximation by averaging. *Siam Journal on Control and Optimization*, 30:838–855, 1992.

B.T. Polyak. Some methods of speeding up the convergence of iteration methods. *Computational Mathematics and Mathematical Physics*, 4(5):1–17, 1964.

A. Ramezani-Kebrya, A. Khisti, and B. Liang. On the stability and convergence of stochastic gradient descent with momentum. *CoRR*, abs/1809.04564, 2018.

T. Schaul, S. Zhang, and Y. LeCun. No more pesky learning rates. In *ICML*, 2013.

M. Schmidt, N. Le Roux, and F. Bach. Minimizing finite sums with the stochastic average gradient. *Mathematical Programming*, 162(1-2):83–112, 2017.

M. Shoeybi, M. Patwary, R. Puri, P. LeGresley, J. Casper, and B. Catanzaro. Megatron-lm: Training multi-billion parameter language models using model parallelism. *CoRR*, abs/1909.08053, 2019.

L. N Smith. Cyclical learning rates for training neural networks. In *WACV*, 2017.

L. N Smith. A disciplined approach to neural network hyper-parameters: Part 1–learning rate, batch size, momentum, and weight decay. *CoRR*, abs/1803.09820, 2018.

L. N. Smith and N. Topin. Super-convergence: Very fast training of residual networks using large learning rates. *CoRR*, abs/1708.07120, 2017.

N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *JMLR*, 15(1):1929–1958, 2014.

T. Tieleman, G. Hinton, et al. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31, 2012.

J. Weston, S. Chopra, and K. Adams. #tagspace: Semantic embeddings from hashtags. In *EMNLP*, 2014.

X. Wu, R. Ward, and L. Bottou. Wngrad: Learn the learning rate in gradient descent. *CoRR*, abs/1803.02865, 2018.

X. Wu, S. S. Du, and R. Ward. Global convergence of adaptive gradient methods for an over-parameterized neural network. *CoRR*, abs/1902.07111, 2019.

T. Yang, Q. Lin, and Z. Li. Unified convergence analysis of stochastic momentum methods for convex and non-convex optimization. *CoRR*, abs/1604.03257, 2016.

Z. Yang, C. Wang, Z. Zhang, and J. Li. Mini-batch algorithms with online step size. *Knowledge-Based Systems*, 165:228–240, 2019.

S. Zagoruyko and N. Komodakis. Wide residual networks. In *BMVC*, 2016.

M. D Zeiler. Adadelta: an adaptive learning rate method. *CoRR*, abs/1212.5701, 2012.

A. Zela, A. Klein, S. Falkner, and F. Hutter. Towards automated deep learning: Efficient joint neural architecture and hyperparameter search. In *ICML AutoML Workshop*, 2018.

G. Zhang, J. Martens, and R. B. Grosse. Fast convergence of natural gradient descent for over-parameterized neural networks. In *NeurIPS*, 2019a.

M. Zhang, J. Lucas, J. Ba, and G. E. Hinton. Lookahead optimizer: k steps forward, 1 step back. In *NeurIPS*, 2019b.