OPTIMIZING THE LEARNING RATE FOR TRAINING NEURAL NETWORKS ON THE FLY

Anonymous authors
Paper under double-blind review

ABSTRACT

Efficient training via gradient-based optimization techniques is an essential building block to the success of artificial neural networks. Extensive research on the impact and the effective estimation of an appropriate learning rate has partly enabled these techniques. Despite the proliferation of data streams generated by IoT devices, digital platforms, etc., previous research has been primarily focused on batch learning, which assumes that all training data is available a priori. However, characteristics such as the gradual emergence of data and the occurrence of distributional shifts also known as *concept drift* pose additional challenges. Therefore, the findings on batch learning may not be applicable to streaming environments, where the underlying model needs to adapt on the fly, each time a new data instance appears. In this work, we seek to address this knowledge gap by (i) evaluating and comparing typical learning rate schedules and optimizers, (ii) exploring adaptations of these techniques, and (iii) providing insights into effective learning rate tuning in the context of stream-based training of neural networks.

1 INTRODUCTION

Artificial neural network models have shown exceptional performance in various domains. One of the main factors leading to such outstanding results is the choice of the optimization method used to train the target model. Almost all modern applications of neural architectures use first-order stochastic optimization methods such as *stochastic gradient descent* (SGD), which iteratively update the parameters of the underlying model based on gradient information. One of the most important variables of such algorithms is the step size or *learning rate* (LR).

As a result, many techniques for setting and optimizing the learning rate have emerged over the years (see Figure 1). For example, based on prior knowledge, the learning rate can be set as a fixed value or as a schedule that changes the step size over time. Alternatively, one could use an adaptive learning rate technique that considers historical gradient information to modify the learning rate at each iteration. In batch learning scenarios, where all training data is available

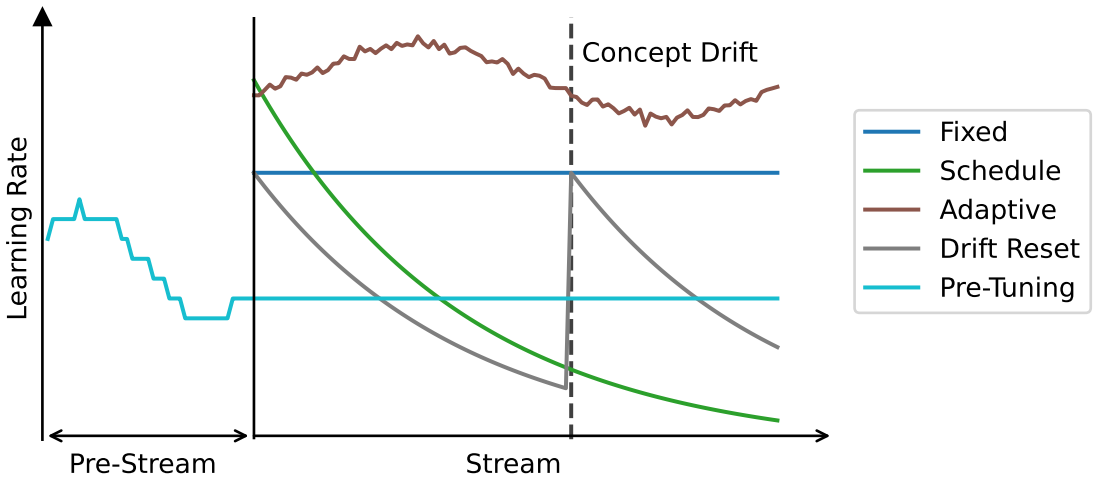


Figure 1: Overview of different learning rate optimization approaches.

a priori, the above methods are well researched. Despite the increasing prevalence of online learning environments, where data becomes available as part of a data stream, their use in such scenarios has received little research attention.

According to Bifet et al. (2010) a machine learning model operating on a data stream must be able to

- R1: process a single instance at a time,
- R2: process each instance in a limited amount of time,
- R3: use a limited amount of memory,
- R4: predict at any time,
- R5: adapt to changes in the data distribution.

These requirements pose additional challenges in selecting an appropriate optimizer and learning rate. To enable more informed decisions when optimizing the learning rate, we provide insight into these challenges and empirically evaluate commonly used optimization techniques in an online learning setting (i). Furthermore, we introduce a *drift reset* mechanism to adapt the learning rate to concept drifts that commonly occur in streaming environments (ii). Finally, we propose a *pre-tuning* approach to make effective use of any *pre-stream* data that may be available prior to stream-based learning for the purpose of learning rate tuning (see Figure 1) (iii).

2 Learning Rate in First-Order Optimization

In the following, we will explain the theoretical background of the learning rate hyperparameter in first-order stochastic optimization. First-order stochastic optimization algorithms, such as stochastic gradient descent, typically aim to solve the following problem

$$\min_{\boldsymbol{\theta}} \mathbb{E}_{\boldsymbol{x}\sim\hat{p}_{\text{data}}}[L(f(\boldsymbol{x};\boldsymbol{\theta}), y)], \tag{1}$$

where L is a loss function that quantifies the prediction error of the model given model parameters $\boldsymbol{\theta}$, data samples $\boldsymbol{x}$ and their corresponding target outputs y, both sampled from the distribution $\hat{p}_{\text{data}}$ defined by a set of training data. The blueprint process for solving this problem using first-order stochastic optimization consists of the following steps for each iteration $t \in 0, \ldots, T$:

1. Draw a mini-batch of samples $\boldsymbol{x}_t$ from the distribution $p(\boldsymbol{x})$.
2. Compute the loss $L_t = L(f(\boldsymbol{x}_t; \boldsymbol{\theta}_t), y_t)$ for examples $\boldsymbol{x}_t$ with their respective labels y_t[1] and current parameters $\boldsymbol{\theta}_t$.
3. Compute gradient $\boldsymbol{g}_t = \nabla_{\boldsymbol{\theta}_t} L_t$ with respect to the parameters.
4. Update the parameters for the next iteration using $\boldsymbol{g}_t$ and possibly information from previous iterations.

For basic SGD, we can define the parameter update performed at the end of each iteration as

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta_t \cdot \boldsymbol{g}_t, \tag{2}$$

where η_t denotes the step size or *learning rate* at timestep t.

The primary trade-off with respect to η is that increasing it speeds up convergence, but also increases stochasticity and the risk of divergence (Bengio, 2012). In fact, Smith & Le (2018), found that the "noise scale" of SGD is tied to η (Smith & Le, 2018).

2.1 Learning Rate Schedules

Often, the performance of a model can be improved by using a schedule that changes the learning rate as training progresses, yielding a vector of step sizes $\boldsymbol{\eta}$ that defines a step size η_t for each timestep $t \in \{0, \ldots, T\}$ (Wu et al., 2019). For example, to ensure fast convergence early in training while mitigating jumping around potential minima later, it is common to use a decaying schedule that starts with a large learning rate and decreases over time. An additional benefit of this approach is potentially better generalization, since larger learning rates can help skip sharp minima with poor generalization (Hochreiter & Schmidhuber, 1997; Chaudhari et al., 2017).

[1]Note that for ease of notation we denote $\boldsymbol{x}_t$ and y_t as vectors and scalars even if they are typically mini-batches of multiple examples.

Commonly used forms of decay are exponential decay, where η_t is calculated as $\eta_t = \eta_1 \cdot \gamma^t$, with $\gamma < 1$, and stepwise decay, which for a regular interval between steps of length s is given as $\eta_1 \cdot \gamma^{\lfloor t/s \rfloor}$. Another common approach is to decay η each time the training loss plateaus for a given number of iterations. Other popular schedules include cyclic learning rates that oscillate η between two values over a predefined interval. For a triangular cycle, the learning rate is computed as

$$\eta_t = \eta_1 + \frac{\hat{\eta} - \eta_1}{2s} \cdot \min_i \{|t - i \cdot s|\}, \tag{3}$$

where $\hat{\eta}$ is the learning rate at the midpoint of each cycle of length s. Some studies (Smith, 2017; Smith & Topin, 2018) have found that cyclic schedules can significantly speed up the convergence of neural networks, in some cases even compared to adaptive techniques like Adam (Kingma & Ba, 2017). While there are many alternatives, in this work we focus on exponential, step, and cyclic learning rates as some of the most commonly used generic schedules. For a comprehensive overview and detailed analysis of learning rate policies, see Wu et al. (2019).

2.2 Adaptive Learning Rates

Several studies have proposed *adaptive optimizers* that increase the robustness of the training process with respect to the learning rate. These optimizers adjust the step size based on previous gradients at each optimization step (Duchi et al., 2011).

Table 1: Overview of additional time- and space-complexity of evaluated adaptive first-order optimizers compared to basic SGD. Values are given in big O notation with respect to the number of model parameters D.

Optimizer	Runtime	Space	Param. specific	LR free
AdaGrad	$\mathcal{O}(5D)$	$\mathcal{O}(1D)$	✓	✗
Adam	$\mathcal{O}(12D)$	$\mathcal{O}(2D)$	✓	✗
WNGrad	$\mathcal{O}(2D)$	$\mathcal{O}(0)$	✗	✗
COCOB	$\mathcal{O}(14D)$	$\mathcal{O}(4D)$	✓	✓
HD[2]	$\mathcal{O}(2D)$	$\mathcal{O}(1D)$	✗	✗
Mechanic[2]	$\mathcal{O}(10D)$	$\mathcal{O}(1D)$	✓	✓
DoG	$\mathcal{O}(5D)$	$\mathcal{O}(1D)$	✗	✓
D-Adapt[2]	$\mathcal{O}(6D)$	$\mathcal{O}(2D)$	✗	✓

One of the earlier techniques in this category is *AdaGrad* (Duchi et al., 2011), which scales the learning rate based on the sum of squares of past gradients for each parameter, resulting in a parameter-specific step size. Several other approaches, such as AdaDelta (Zeiler, 2012) and RMSProp, later built on AdaGrad's scaling approach. The same is true for the widely used Adam optimizer (Kingma & Ba, 2017), which adds a momentum term from prior gradients to speed up convergence for parameters with consistent derivatives. Another AdaGrad based optimizer is *WNGrad* (Wu et al., 2020), which adaptively scales each parameter update based on the squared sum of past gradients.

So-called *parameter-free* gradient-based optimization approaches aim to eliminate the learning rate altogether by optimizing it as training progresses. For example, the *COCOB* algorithm (Orabona & Tommasi, 2017) models parameter optimization as a gambling problem, where the goal is to maximize the reward from betting on each gradient. The resulting strategy is equivalent to running a meta-optimization algorithm that estimates the expected optimal learning rate (Orabona & Tommasi, 2017). Several other studies (van Erven & Koolen, 2016; Baydin et al., 2018; Cutkosky et al., 2023) have also used the idea of learning η via a meta-optimization process. The *hypergradient descent* (HD) approach (Baydin et al., 2018), for example, adapts the learning rate of a base optimizer like SGD using a meta-gradient descent procedure, although this does not remove the learning rate entirely, but replaces it with a less sensitive hypergradient step size. Mechanic (Cutkosky et al., 2023) pursues the same goal by applying a meta *online convex optimization* (OCO) algorithm to an arbitrary base optimizer.

Research has shown that in an OCO problem setting with stationary data, the worst-case optimal fixed step size for SGD is

$$\eta^* = \frac{||\boldsymbol{\theta}_1 - \boldsymbol{\theta}^*||}{\sqrt{\sum_{t=1}^{n} ||\boldsymbol{g}_t||^2}}. \tag{4}$$

Multiple parameter-free optimizers, make use of this notion. As its name suggests, the *Distance over Gradients* (DoG) (Ivgi et al., 2023) algorithm estimates the unknown numerator in Equation 4 as the maximum distance $\max_{i<t} ||\boldsymbol{\theta}_1 - \boldsymbol{\theta}_i||$ between the initial parameters and the parameters of all previous iterations. DoG additionally

[2]Variant with SGD as the underlying optimizer.

uses polynomial decay averaging as proposed by Shamir & Zhang (2012). *D-Adaptation* by Defazio & Mishchenko (2023), on the other hand, employs weighted dual averaging (Duchi et al., 2012) to compute bounds on the distance between initial and optimal parameters. Although adaptive optimization techniques seem intuitively well suited for non-stationary data, their application to data streams has rarely been investigated. Therefore, we assess the suitability of some of the most prominent adaptive optimizers, listed in Table 1, for stream-based learning.

3 Learning Rate in Online Learning

In a batch learning setting, optimizing the learning rate involves minimizing the expected loss on a hold-out set of validation data. Formally, we can express this task as

$$\min_{\boldsymbol{\eta}} \mathbb{E}_{(\boldsymbol{x},y)\sim\hat{p}_{\text{val}}} L(f(\boldsymbol{x};\boldsymbol{\theta}_n), y), \tag{5}$$

where samples $(\boldsymbol{x}, y)$ are drawn from a distribution $\hat{p}_{\text{val}}$ defined by a validation dataset and $\boldsymbol{\theta}_n$ are the model parameters at the end of training that depend on the learning rate schedule $\boldsymbol{\eta}$. In online learning where data is generated incrementally, this notion of learning rate optimization is not feasible. Due to requirements **R1-R5**, models operating in an online learning environment should be evaluated in a *prequential* manner (Bifet et al., 2010), where each sample $\boldsymbol{x}_t$ in the data stream is first used to test and then to train the model, ensuring that testing is done on previously unobserved data.

Training in such a scenario can be more accurately modeled as an online convex optimization problem (Shalev-Shwartz, 2011; Hazan, 2016), where the optimizer suffers a loss $L(f(\boldsymbol{x};\boldsymbol{\theta}_t), y)$ and produces updated parameters $\boldsymbol{\theta}_{t+1}$ at each iteration of the data stream. Learning rate optimization in this setting can be formulated as

$$\min_{\boldsymbol{\eta}} \sum_{t=1}^{n} \mathbb{E}_{(\boldsymbol{x},y)\sim\hat{p}^{(t)}_{\text{stream}}} L(f(\boldsymbol{x};\boldsymbol{\theta}_t), y). \tag{6}$$

Compared to Equation 5, Equation 6 features some key differences. Due to the requirement to be able to predict at any time (**R4**), the goal is to minimize the expected total sum of losses over all timesteps of the prequential evaluation process, instead of the validation loss for the final parameters $\boldsymbol{\theta}_n$. Therefore, the speed of convergence is more important in the streaming setting, while the performance of the final $\boldsymbol{\theta}_n$ parameter has a much smaller impact. Since memory is limited (Requirement 1), it is also not possible to continue training on previously observed data as long as the loss decreases, which puts even more emphasis on fast adaptation. At the same time, a higher learning rate that temporarily increases the loss by skipping local minima may be suboptimal with respect to Equation 6, even if it eventually leads to a lower loss. Another difference to conventional batch learning is that the data distribution $\hat{p}^{(t)}_{\text{stream}}$ is time-dependent, due to the fact that streams are often subject to distributional changes such as *concept drifts* over time (Widmer & Kubat, 1996). While other forms of drifts exist, we use *concept drift* as an umbrella term for any kind of distributional shift in the following.

3.1 Learning Rate Tuning

Although strict online learning assumes that no data is available prior to the deployment of a model in the streaming environment, smaller amounts of data may be available prior to the stream learning process in practical scenarios. Such data can then be used to tune the learning rate of the model before stream deployment. However, the differences in evaluation schemes described above may cause conventional learning rate tuning to produce poor results for stream-based learning. In offline learning, learning rate tuning is typically performed by repeating the training process for a grid of possible parameter values and selecting the value that yields the best target metric on a held-out validation set after the training is completed. As a result of only considering the performance at the end and not throughout the training process, this tuning approach may result in poor results with respect to the goal of learning rate optimization in streaming environments (see Equation 6). Therefore, we propose the modified tuning approach described in Algorithm 1 aimed at approximating Equation 6, which we call learning rate *pre-tuning*.

To emulate the data stream to be processed after tuning, we continuously draw samples with replacement from the tuning data $\mathbb{X}$ in a bootstrapping procedure instead of training on all data for multiple epochs. By doing so, we aim to increase the variability of the data and thus the similarity to a real data stream. We then optimize the learning rate and any optional learning rate schedule parameters γ with respect to the selected performance metric m over the emulated stream $\mathbb{S}$. While we use a simple grid search for this purpose in algorithm 1, any parameter search technique could be used here. We provide a detailed experimental evaluation of our approach in Section 4.

Algorithm 1 Learning Rate Pre-Tuning

Require: Set of learning rate values $\mathbb{H}$, set of schedule parameter values $\mathbb{G}$, data samples $\mathbb{X}$
Require: Optimization function o, metric function to tune m, number of tuning steps n_{steps}

$\mathbb{S} \leftarrow \{(\boldsymbol{x}_i, y_i) \sim \hat{p}_{\mathbb{X}} | \forall i \in \{1, \ldots, n_{\text{steps}}\}\}$ ▷ Create artificial stream by sampling $\mathbb{X}$ with replacement.
$v^* \leftarrow -\infty$
for η **in** $\mathbb{H}$, γ **in** $\mathbb{G}$ **do**
  $v \leftarrow v_{\text{init}}$ ▷ Initialize metric value.
  for $\boldsymbol{x}, y$ **in** $\mathbb{S}$ **do**
    $\hat{y} \leftarrow f(\boldsymbol{x}, \boldsymbol{\theta})$ ▷ Calculate predictions with model function f.
    $v \leftarrow m(\hat{y}, y, v)$ ▷ Update metric value.
    $\boldsymbol{g} \leftarrow \nabla_{\boldsymbol{\theta}} L(\hat{y}, y)$ ▷ Calculate gradient of predictive loss w.r.t. $\boldsymbol{\theta}$.
    $\boldsymbol{\theta}, \eta \leftarrow o(\boldsymbol{\theta}, \boldsymbol{g}, \eta, \gamma)$ ▷ Update model parameters $\boldsymbol{\theta}$ and learning rate η.
  end for
  if $v > v^*$ **then**
    $\eta^* \leftarrow \eta$ ▷ Update best learning rate.
    $\gamma^* \leftarrow \gamma$ ▷ Update best schedule parameters.
  end if
end for
return η^*, γ^* ▷ Return best hyperparameter values.

3.2 Concept Drift Adaptation

Concept drift requires repeated adaptation of the model parameters. If post-drift training is interpreted as a new online optimization problem, the worst-case optimal learning rate can be computed according to Equation 4, replacing the initial parameter values $\boldsymbol{\theta}_1$ with the values at the time of drift onset $\boldsymbol{\theta}_{t_{\text{drift}}}$. As a result, stronger drifts that cause $\boldsymbol{\theta}^*$ to move away from $\boldsymbol{\theta}_{t_{\text{drift}}}$ can benefit from larger learning rates.

Based on this notion, we propose a simple adaptation to decaying learning rate schedules that resets η to its original value when a concept drift is detected. An exponential schedule modified with our approach will thus yield learning rates of

$$\eta_t = \eta_1 \cdot \gamma^{t - t_{\text{drift}}}, \tag{7}$$

where t_{drift} marks the timestep at which the drift was last detected. For drift detection, we apply ADWIN (Bifet & Gavaldà, 2007) to the prequential losses. To avoid mistakenly detecting loss decreases as concept drift, we use a one-sided ADWIN variant that tests only for increases.

Our approach is similar to some *forgetting mechanisms* commonly used in conventional online learning (Gama et al., 2014). To improve model plasticity, such mechanisms partially or completely reset the current model parameters to their initial values. However, we hypothesize that this approach is not well suited for neural-based approaches. The reason is that, under the assumption of convexity, the newly initiated parameters must be closer to the optimal parameters $\boldsymbol{\theta}^*$ than the current parameters to be beneficial. We experimentally compare our approach with this weight-reset mechanism in Section 4.

4 Experiments

We empirically evaluate our hypotheses using the following setup[3]:

We use both synthetic and publicly available real-world classification datasets with different sizes and types of concept drift, listed in Table 2. Our evaluations include two synthetic *Random Radial Basis Function* (RBF) datasets that we manipulated to incorporate concept drift using the online learning framework *River* (Montiel et al., 2020).

We further employ the *Electricity* and *Covertype* (Blackard, 1998) datasets, which are commonly used to evaluate online learning models, as well as two *Insects* datasets (Souza et al., 2020) with predefined types of concept drift. *Covertype* is available on *OpenML* (Vanschoren et al., 2014), while the remaining datasets are part of River. We limit our investigations to multi-layer perceptrons (MLPs) with a single-hidden-layer and hidden units matching the number of input features implemented with *PyTorch* (Paszke et al., 2019). Even though more elaborate architectures could in

[3]Code available at `anonymous.4open.science/r/LODL-D458/`.
[4]For Covertype we use only the first 100,000 from a total of 581,012 instances.

Table 2: Datasets used for experimental evaluations.

Type	Data Stream	Instances	Features	Classes
Synth.	RBF abrupt	100,000	20	5
	RBF incremental	100,000	20	5
Real	Insects abrupt	52,848	33	6
	Insects gradual	24,150	33	6
	Covertype	100,000[4]	54	7
	Electricity	45,312	8	2

theory produce better results, they also typically require more tuning and longer training than smaller models making them less suited for streaming environments from a practical standpoint.

We tune the base learning rate η_1 of all but the parameter-free approaches using a grid search of ten geometrically spaced values and configure adaptive optimizers with their default parameter values. For HD, Mechanic and D-Adaptation we select SGD as the base algorithm. We select a fixed factor γ for decay schedules on all datasets. For the proposed learning rate resetting mechanism, we select a smaller decay factor and set the confidence level δ for drift detection to 10^{-4}. For our evaluations we process each dataset sequentially, emulating streams of mini-batches of four instances each, while recording the prequential accuracy and other metrics in intervals of 25 iterations. We report our results averaged over five random seeds.

4.1 Learning Rate Schedules

To evaluate the effectiveness of our learning rate resetting mechanism for drift adaptation (see Equation 3.2), we compare its average prequential accuracy to that of model weight resetting, commonly used in online learning.

Table 3: Average prequential accuracy [%] for static and drift adaptive learning rate schedules with SGD. For LR-Reset Oracle we manually reset the learning rate at timesteps where concept drift occurs. Best values are shown in **bold**, values within the 1σ interval of best values underlined.

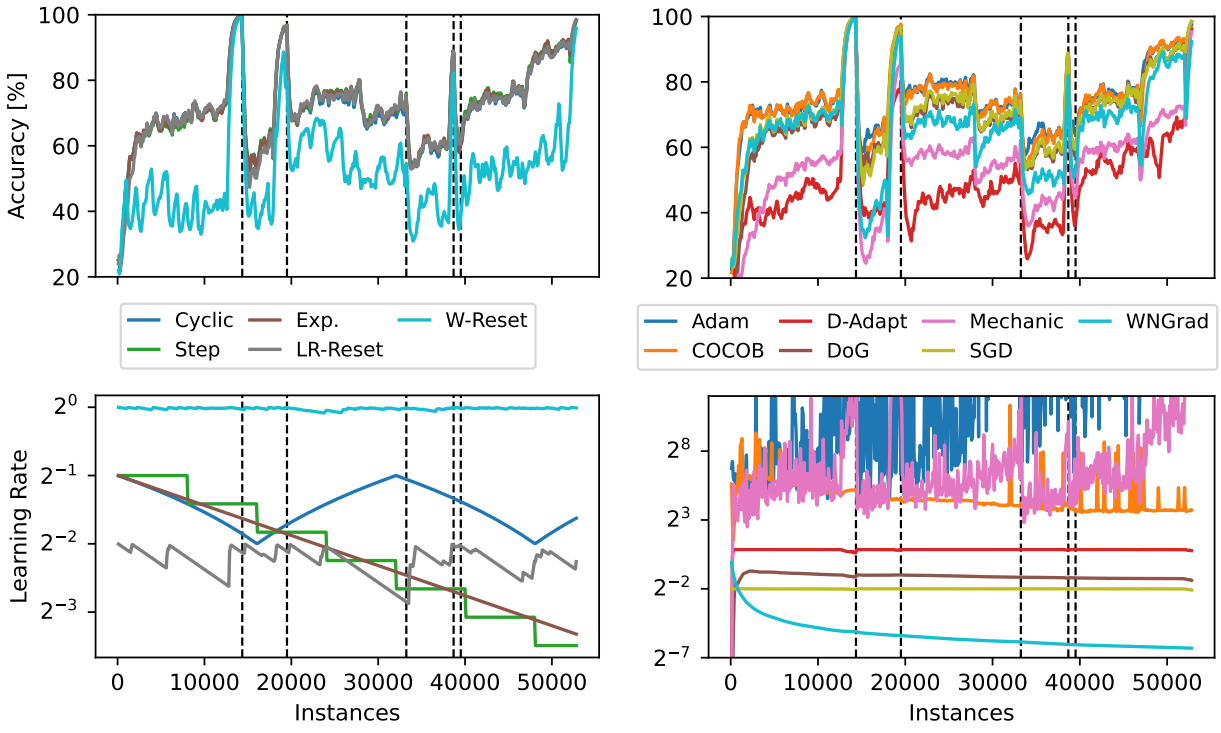
Schedule	RBF abrupt	RBF incr.	Covertype	Electricity	Insects abrupt	Insects gradual
Fixed	94.79±.32	70.95±2.89	83.42±.50	73.77±.40	71.50±.08	75.31±.21
Step	**94.87±.28**	70.19±3.02	82.89±.37	73.62±.53	**72.23±.27**	75.83±.21
Cyclic	94.79±.32	**74.96±.86**	**83.44±.08**	68.38±.81	71.74±.39	75.64±.06
Exponential	94.85±.29	70.23±2.40	82.95±.26	73.51±.48	72.19±.37	**75.91±.14**
Weight-Reset	69.96±.38	65.13±.80	83.12±.13	70.08±1.66	51.52±.90	62.55±2.34
LR-Reset (Ours)	94.83±.26	73.38±2.32	82.99±.20	**73.79±.62**	71.73±.20	75.52±.12
LR-Reset Oracle	95.12±.21	—	—	—	71.88±.26	—

As can be seen in Table 3, our approach clearly outperforms weight-resetting on all but *Covertype*, but rarely yields an advantage over a static schedule. Since the oracle variant of our resetting approach, that was triggered only for timesteps with drift, shows only marginally better results, this performance gap is not caused by the drift detector. Rather, it appears that using a larger initial learning rate and slower decay is sufficient to ensure adequate adaptability to concept drift throughout a data stream, while providing better stability in later stages. Overall, a slower but static decay paired with a larger initial learning rate seems to be preferable to a more aggressive schedule with our drift resetting mechanism, unless severe concept drift as in *RBF incremental* is expected.

With accuracy values within 1σ of each other on all evaluated streams, the stepwise decay shows almost identical performance to the exponential decay. The accuracy of the cyclic schedule for *RBF incremental* and *Covertype*, on the other hand, significantly outperforms the other static and adaptive schedules, but lags behind on all other streams. We also did not find an order of magnitude improvement in convergence speed as observed by (Smith & Topin, 2018) for the scenario studied.

4.2 Adaptive Learning Rates

Our results for adaptive optimizers displayed in Table 4, show a strong data dependency as none of the evaluated algorithms significantly outperforms its competitors on average. However, since SGD yields the best accuracy on *RBF abrupt* but is clearly surpassed on *Insects abrupt*, the type of concept drift does not seem to be significant. Due to its



(a) Schedules

(b) Adaptive Optimizers

Figure 2: Prequential accuracy and learning rate for different schedules and adaptive optimizers on *Insects abrupt* dataset. Concept drifts are marked by dashed lines. Accuracy is exponentially smoothed with a decay factor of 0.75.

simplicity and favorable computational efficiency, it appears that SGD should be selected out of the non-parameter-free approaches in most cases. The SGD variant of Hypergradient Descent (HD) (Baydin et al., 2018) and WNGrad (Wu et al., 2020) on the other hand seem to rarely be optimal choices.

Table 4: Average prequential accuracy [%] for different optimizers. Best values are shown in **bold**, values within the 1σ interval of best values underlined.

	Optimizer	RBF abrupt	RBF incr.	Covertype	Electricity	Insects abrupt	Insects gradual
Tuned	SGD	**94.79±.32**	70.95±2.89	**83.42±.50**	73.77±.40	71.50±.08	75.31±.21
	Adam	93.45±.30	69.26±5.14	79.01±.27	69.79±.54	**75.38±.24**	75.78±.74
	AdaGrad	92.45±1.37	52.87±6.62	81.68±.35	**76.99±1.20**	74.87±.40	**77.15±.27**
	WNGrad	87.30±.68	44.92±.73	76.98±.15	70.80±.59	66.25±.19	66.75±.40
	HD	93.92±.31	**72.29±2.90**	83.36±.25	73.83±.32	70.67±.06	73.37±.21
LR-Free	COCOB	**93.40±.38**	63.52±2.70	82.27±.46	**84.30±.56**	**74.75±.11**	**77.00±.05**
	DoG	92.72±.59	**73.17±2.72**	**83.07±.64**	71.53±.70	70.59±.26	74.01±.21
	D-Adapt	74.91±4.22	45.47±2.75	76.69±.79	66.03±1.75	50.05±11.26	48.21±10.62
	Mechanic	88.94±.58	49.26±1.44	78.67±.18	50.73±7.60	55.31±21.47	65.80±.53

In the category of learning rate free optimizers COCOB (Orabona & Tommasi, 2017), outperformed its competitors on all but two datasets. It comes close to or even exceeds the best tuned approaches in terms of accuracy. Although yielding lower accuracy on average, DoG also comes within reach of the tuned methods, while offering much better runtime and memory efficiency compared to COCOB (see Table 1). Mechanic (Cutkosky et al., 2023) and D-Adaptation (Defazio & Mishchenko, 2023) performed significantly worse than their competitors on the evaluated streams.

The learning rate curves shown in Figure 2a, provide an indication regarding the reason for the poor performance of WNGrad and D-Adaption. Whereas the learning rate of DoG quickly approaches the optimal learning rate, WNGrad and D-Adaptation diverge considerably from it.

The learning rate of the best performing Adam exhibits spikes for most change points, suggesting some form of adaptability to drift. However, since the much worse performing Mechanic shows similar spikes, this is unlikely to be contributing significantly to Adam's high accuracy on *Insects abrupt*. Instead, it likely stems from its second moment scaling, which is also a feature of the similarly performing AdaGrad.

It may also be noted that the learning rates of the COCOB, Adam and Mechanic optimizers with parameter-specific learning rates exceed those of single value step sizes by multiple orders of magnitude. This is an effect of second moment scaling, which creates larger learning rates for parameters with small and consistent gradients (Cutkosky et al., 2023). Therefore, the norms of parameter updates generated by these approaches are not necessarily larger.

4.3 Learning Rate Tuning

We evaluate our pre-tuning approach by using either 500 or 1000 instances held out at the beginning of each stream for tuning. We assess Multi-Layer Perceptrons (MLPs) with 64 or 128 hidden units per layer and either one or three hidden layers. Although our technique consistently performed well across all architectures, we specifically highlight the smallest network as it best represents resource-critical streaming applications. For an overview of results across all architectures, please refer to our supplementary material.

To determine the learning rate and decay factor, we rely on the optimal mean prequential accuracy. This choice accounts for the potential bias towards learning rates associated with lower initial losses, as loss values often decrease notably during training.

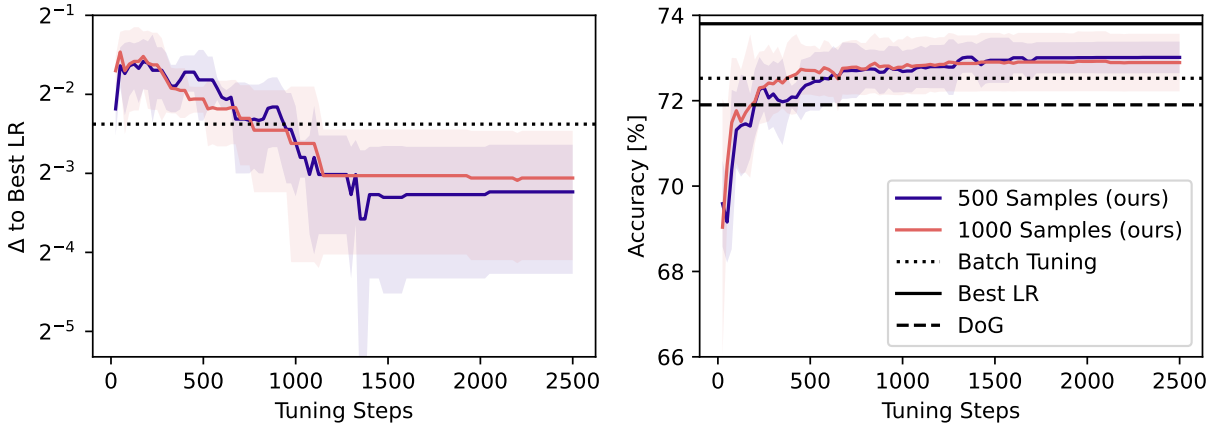


Figure 3: Absolute difference between pre-tuned and optimal learning rate and resulting accuracy on data streams for SGD and an exponential learning rate schedule with 500 or 1000 tuning samples. DoG is not included in first subplot since it is not intended for use with LR decay. Results are averaged over all real-world datasets. The shaded area represents the 1σ-interval.

Figure 3 shows the absolute difference between the learning rate resulting from the pre-tuning process and the optimal value $|\eta_p - \eta^*|$ at each tuning step averaged over all real-world datasets. Batch tuning with 800 training and 200 validation samples initially yields a better approximation of the optimal learning rate. However, our streaming-specific approach undercuts the baseline after 1000 tuning steps, consistently decreasing and ultimately reaching approximately half of the approximation error observed in batch tuning. The performance also remains nearly identical, even when using only 500 samples for tuning, which demonstrates the data efficiency of pre-tuning. The superior performance of our approach is also reflected in the accuracy scores depicted in the right subfigure of Figure 3. In fewer than 1000 steps, our approach comes closer to the optimal learning rate and also exceeds the accuracy of conventional tuning. Notably, our approaches emulation of the online learning process, considering the average accuracy throughout an emulated stream, sets it apart from conventional tuning, which solely focuses on performance at the end of the tuning process. Pre-Tuning also surpasses DoG, which we selected DoG as a baseline due to being the best performing parameter-free technique within the group of optimizers with a global learning rate that is most comparable to our approach.

In conclusion, our proposed tuning approach enables significantly better learning rate selection for prequential evaluation on data streams compared to both conventional tuning and DoG. Additionally, pre-tuning has the benefit that once

completed, no additional memory or runtime costs are incurred. In streaming applications, where computing resources are often times a limiting factor, this could be a critical advantage. However, if computational efficiency is insignificant, the highly performant but expensive COCOB (Orabona & Tommasi, 2017) or the slightly less performant and much less expensive DoG (Ivgi et al., 2023) may be more appropriate. The usefulness of our pre-tuning approach is also likely to be limited in cases where the available tuning data is not representative for the subsequent data stream, which would for example be the case for data streams with extreme concept drift. In such cases, approaches with a dynamic learning rate are presumably superior.

5 Conclusion

In this work, we investigate the influence and selection of the learning rate and optimization procedure with respect to training neural networks in streaming environments. We first provide theoretical background on discrepancies between learning rate optimization in conventional batch learning and online learning. Based on these differences, we derive a simple mechanism resetting the learning rate on concept drift occurrences. We then give an overview learning rate free algorithms popular in batch learning, which we compare experimentally on multiple synthetic and real-world datasets, finding both COCOB and DoG to come close to the performance of optimizers with tuned learning rates. Lastly, we introduce a streaming specific learning rate tuning approach that grants significant performance increases over conventional tuning via a train-validation split.

References

Atilim Gunes Baydin, Robert Cornish, David Martinez Rubio, Mark Schmidt, and Frank Wood. Online Learning Rate Adaptation with Hypergradient Descent. In *ICLR Proceedings*, February 2018.

Yoshua Bengio. Practical recommendations for gradient-based training of deep architectures, September 2012.

Albert Bifet and Ricard Gavaldà. Learning from Time-Changing Data with Adaptive Windowing. In *Proceedings of the 2007 SIAM International Conference on Data Mining*, pp. 443–448. Society for Industrial and Applied Mathematics, April 2007. ISBN 978-0-89871-630-6 978-1-61197-277-1. doi: 10.1137/1.9781611972771.42.

Albert Bifet, Geoffrey Holmes, Richard Kirkby, and Bernhard Pfahringer. MOA: Massive online analysis. *Journal of Machine Learning Research*, 11, May 2010.

Jock Blackard. Covertype. UCI Machine Learning Repository, 1998.

Pratik Chaudhari, Anna Choromanska, Stefano Soatto, Yann LeCun, Carlo Baldassi, Christian Borgs, Jennifer Chayes, Levent Sagun, and Riccardo Zecchina. Entropy-SGD: Biasing Gradient Descent Into Wide Valleys, April 2017.

Ashok Cutkosky, Aaron Defazio, and Harsh Mehta. Mechanic: A Learning Rate Tuner, June 2023. URL http://arxiv.org/abs/2306.00144.

Aaron Defazio and Konstantin Mishchenko. Learning-Rate-Free Learning by D-Adaptation, May 2023. URL http://arxiv.org/abs/2301.07733.

J. C. Duchi, A. Agarwal, and M. J. Wainwright. Dual Averaging for Distributed Optimization: Convergence Analysis and Network Scaling. *IEEE Transactions on Automatic Control*, 57(3):592–606, March 2012. ISSN 0018-9286, 1558-2523. doi: 10.1109/TAC.2011.2161027.

John C. Duchi, Elad Hazan, and Yoram Singer. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *Journal of Machine Learning Research*, 12(61):2121–2159, February 2011.

João Gama, Indrė Žliobaitė, Albert Bifet, Mykola Pechenizkiy, and Abdelhamid Bouchachia. A survey on concept drift adaptation. *ACM Computing Surveys*, 46(4):1–37, April 2014. ISSN 0360-0300, 1557-7341. doi: 10.1145/2523813.

Elad Hazan. Introduction to Online Convex Optimization. *Foundations and Trends® in Optimization*, 2(3-4):157–325, August 2016. ISSN 2167-3888, 2167-3918. doi: 10.1561/2400000013.

Sepp Hochreiter and Jürgen Schmidhuber. Flat Minima. *Neural Computation*, 9(1):1–42, January 1997. ISSN 0899-7667, 1530-888X. doi: 10.1162/neco.1997.9.1.1.

Maor Ivgi, Oliver Hinder, and Yair Carmon. DoG is SGD's Best Friend: A Parameter-Free Dynamic Step Size Schedule, July 2023. URL http://arxiv.org/abs/2302.12022.

Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization, January 2017.

Jacob Montiel, Max Halford, Saulo Martiello Mastelini, Geoffrey Bolmier, Raphael Sourty, Robin Vaysse, Adil Zouitine, Heitor Murilo Gomes, Jesse Read, Talel Abdessalem, and Albert Bifet. River: Machine learning for streaming data in Python, December 2020.

Francesco Orabona and Tatiana Tommasi. Training Deep Networks without Learning Rates Through Coin Betting. In *NIPS*, 2017.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (eds.), *NeurIPS Proceedings*, pp. 8024–8035. Curran Associates, Inc., 2019. URL http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

Shai Shalev-Shwartz. Online Learning and Online Convex Optimization. *Foundations and Trends® in Machine Learning*, 4(2):107–194, 2011. ISSN 1935-8237, 1935-8245. doi: 10.1561/2200000018.

Ohad Shamir and Tong Zhang. Stochastic Gradient Descent for Non-smooth Optimization: Convergence Results and Optimal Averaging Schemes, December 2012. URL http://arxiv.org/abs/1212.1824.

Leslie N. Smith. Cyclical Learning Rates for Training Neural Networks, April 2017.

Leslie N. Smith and Nicholay Topin. Super-Convergence: Very Fast Training of Neural Networks Using Large Learning Rates, May 2018. URL http://arxiv.org/abs/1708.07120.

Samuel L. Smith and Quoc V. Le. A Bayesian Perspective on Generalization and Stochastic Gradient Descent, February 2018.

Vinicius M. A. Souza, Denis M. dos Reis, Andre G. Maletzke, and Gustavo E. A. P. A. Batista. Challenges in Benchmarking Stream Learning Algorithms with Real-world Data. *Data Mining and Knowledge Discovery*, 34(6): 1805–1858, November 2020. ISSN 1384-5810, 1573-756X. doi: 10.1007/s10618-020-00698-5.

Tim van Erven and Wouter M. Koolen. MetaGrad: Multiple Learning Rates in Online Learning, November 2016.

Joaquin Vanschoren, Jan N. van Rijn, Bernd Bischl, and Luis Torgo. OpenML: Networked science in machine learning. *ACM SIGKDD Explorations Newsletter*, 15(2):49–60, June 2014. ISSN 1931-0145, 1931-0153. doi: 10.1145/2641190.2641198.

Gerhard Widmer and Miroslav Kubat. Learning in the presence of concept drift and hidden contexts. *Machine Learning*, 23(1):69–101, April 1996. ISSN 1573-0565. doi: 10.1007/BF00116900.

Xiaoxia Wu, Rachel Ward, and Léon Bottou. WNGrad: Learn the Learning Rate in Gradient Descent, November 2020.

Yanzhao Wu, Ling Liu, Juhyun Bae, Ka-Ho Chow, Arun Iyengar, Calton Pu, Wenqi Wei, Lei Yu, and Qi Zhang. Demystifying Learning Rate Policies for High Accuracy Training of Deep Neural Networks, October 2019. URL http://arxiv.org/abs/1908.06477.

Matthew D. Zeiler. ADADELTA: An Adaptive Learning Rate Method, December 2012.

A Appendix

You may include other additional sections here.