Learning Rate Optimization for Online Deep Learning

Anonymous submission

Abstract

Efficient training via gradient-based optimization techniques is an essential building block to the success of deep learning. 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 and non-stationarity of data pose additional challenges. Therefore, the findings on batch learning may not be applicable to deep learning in streaming environments. 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 deep learning.

1 Introduction

Deep learning models have demonstrated exceptional performance in various domains, with the choice of the optimization optimizer playing a crucial role in achieving outstanding results. In the context of batch learning, where all data is available at a time, extensive research has been conducted to explore optimization techniques for deep learning architectures and numerous methods have emerged to effectively update the weights of these architectures. Further, the recent development of deep learning frameworks for online learning scenarios foster the application and the research of deep learning models in dynamic scenarios such as on data streams. However, with the application of deep learning models on data streams new challenges arise, as data streams evolve over time the models are affected by potential changes in the underlying data structure, that is also referred to as concept drift. In order to achieve high predictive performance and a fast adaptation of the networks weights to new data patterns a suitable choice of the underlying optimizer becomes crucial. This paper aims to bridge this knowledge gap by investigating how the choice of optimizer changes when transitioning from batch learning to online learning scenarios and discusses different optimization strategies when applying deep learning models in online learning scenarios. Specifically, we address the following research questions:

1. How does the choice for the optimizer change from batch to online learning?
2. What are practical choices for gradient-based online training of deep architectures in online learning?
3. Are adaptive optimization methods better suited in Online Deep Learning?

Online Learning Req. in K text einbe ten!

According to Bifet et al. (2010) a machine learning model operating in such an environment 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.

2 Learning Rate in First Order Stochastic Optimization

In the following, we will explain the theoretical background of first-order stochastic optimization enabling modern deep learning models. We will also outline the differences between the application of these techniques in traditional batch learning and online learning in terms of impact of the learning rate and its optimization.

First-order stochastic optimization algorithms like stochastic gradient descent typically aim to solve

$$\min_{\theta} \mathbb{E}_{x \sim p(x)}[\mathcal{L}(x, \theta)], \tag{1}$$

where $\mathcal{L}(x, \theta)$ represents a loss function that quantifies the predictive error of the model given a mini-batch of data samples x and model parameters θ. The blueprint process of solving this problem via first order stochastic optimization consists of the following steps for each iteration $t \in 0, \ldots, T$:

1. Draw a mini-batch of samples x_t from distribution $p(x)$.
2. Calculate the loss $\mathcal{L}_t = \mathcal{L}(x_t, \theta_t)$ for x_t and current parameters θ_t.
3. Compute the gradient $g_t = \nabla_{\theta_t} \mathcal{L}_t$ with respect to the parameters.
4. Update the parameters for the next iteration using g_t and potentially information from past iterations.

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

$$\theta_t = \theta_t - \eta_t \cdot g_t, \tag{2}$$

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

As previously described, the learning rate has an immense impact on the performance of the optimization process and therefore on the performance of a deep learning model as a whole.

The primary trade-off to consider with respect to the choice of η is that increasing the learning rate speeds up convergence, but at the same time also increases stochasticity and therefore leads to the divergence of the training criterion beyond a certain threshold. (Bengio 2012). In fact, Smith and Le (2018), found that when modelling SGD as a stochastic differential equation, the "noise scale" is directly tied to η (Smith and Le 2018). In biological terms, increasing the learning rate increases plasticity, whereas decreasing it increases stability.

2.1 Learning Rate Schedules

While using a single fixed learning rate $\eta_t = \eta$ for all iterations simplifies the learning selection and can often yield sufficient performance, results can generally be improved with a schedule with step sizes specific to each iteration (Wu et al. 2019). To ensure fast convergence at the start of training, while mitigating jumping around potential minima at later stages it is, for instance, common to use a decaying schedule starting with a large learning rate that decreases over time. An additional benefit of this approach is that of potentially better generalization, since larger learning rates can help skipping over sharp minima with poor generalization (Hochreiter and Schmidhuber 1997; Chaudhari et al. 2017). Some have likened this procedure to simulated annealing, which shifts its focus from exploration at high temperatures to exploitation once temperatures have sufficiently decreased (Smith et al. 2018).

A commonly used forms of decay are exponential decay, where η_t calculates as

$$\eta_t = \eta_0 \cdot \gamma^t, \tag{3}$$

with $\gamma < 1$, and stepwise decay, which for a regular interval between steps of length s is given as

$$\eta_t = \eta_0 \cdot \gamma^{\lfloor t/s \rfloor}. \tag{4}$$

Other popular options include cyclic learning rate schedules which oscillate η between two values over a predefined interval. For a basic triangular cycle, the learning rate calculates as

$$\eta_t = \eta_0 + \frac{\hat{\eta} - \eta_0}{2s} \cdot \min_i \{|t - i \cdot s|\}, \tag{5}$$

with $\hat{\eta}$ being the learning rate at the middle of each cycle of length s. Some studies (Smith 2017; Smith and Topin 2018) have found cyclic schedules to significantly speed up the convergence of neural networks even when compared to adaptive techniques like Adam in some cases (Kingma and Ba 2017). While there are many other learning rate schedules, we focus on the use of the three aforementioned schedules within data streaming applications in this work. For a comprehensive overview and detailed analysis on learning rate policies, we refer to Wu et al. (2019).

2.2 Adaptive Learning Rates

While determining the learning rate through a separate tuning phase with parameter searches like grid- or random-search is still the de facto standard in deep learning (Defazio and Mishchenko 2023), this approach causes significant computational overhead.

To reduce this overhead, several previous works have developed *adaptive optimizers*, which adjust the learning rate based on additional information about the loss landscape obtained from previous gradients at each optimization step, increasing the robustness with respect to the step size (Duchi, Hazan, and Singer 2011).

One of the earlier optimizers in this category is *AdaGrad* (Duchi, Hazan, and Singer 2011), which divides the learning rate by the square root of the uncentered total sum of squares over all previous gradients, for each model parameter resulting in a parameter specific learning rate. Unlike a single global value, parameter specific learning rates therefore not only influence the length, but also the direction of update steps, in case of AdaGrad by shifting updates in the direction of smaller gradients (Wu, Ward, and Bottou 2020).

Among several other approaches like AdaDelta (Zeiler 2012, see e.g.) and RMSProp (Tieleman and Hinton 2012), Kingma and Ba (2017) subsequently introduced Adam as an extension of AdaGrad, that additionally takes a momentum term of past gradients into account (Sutskever et al. 2013, see) to speed up the convergence for parameters with consistent gradients.

Another optimizer building on AdaGrad is *WNGrad* (Wu, Ward, and Bottou 2020), which adaptively scales each parameter update based on the squared sum of past gradients. By doing so, WNGrad achieves a single, step size robust, learning rate (Wu, Ward, and Bottou 2020).

While adaptive approaches such as AdaGrad and Adam have been shown to reduce the dependence on the learning rate, they often times still require manual tuning (Wu, Ward, and Bottou 2020). A problem that parameter-free variants of SGD aim to solve by estimating the optimal step size online as training progresses, thus eliminating the learning rate altogether.

In one of the earlier works on parameter-free optimization, Schaul, Zhang, and LeCun (2013) proposed *vSGD*, which, like Adam, uses first and second order moments of the gradients as well as local curvature information to estimate η (Schaul, Zhang, and LeCun 2013). The authors obtain the latter by estimating positive diagonal entries of the Hessian with respect to the parameters through a back-propagation formula (Schaul, Zhang, and LeCun 2013). Due to the age and higher complexity compared to similar approaches, we did not include vSGD in our evaluations.

Instead of using curvature information for adapting η, the *COCOB* algorithm proposed by Orabona and Tommasi

(2017) models parameter optimization as a gambling problem, in which the goal is to maximize the rewards obtained from betting on each gradient. The model parameters are then computed based on the rewards accumulated over all previous timesteps (Orabona and Tommasi 2017). Intuitively, this corresponds to running a meta optimization algorithm, that estimates the expected optimal learning rate in parallel with the actual parameter optimization process.

Several other contributions (van Erven and Koolen 2016; Baydin et al. 2018; Cutkosky, Defazio, and Mehta 2023) have also used the idea of learning η via a meta-optimization process. The *hypergradient descent* (HD) approach (Baydin et al. 2018) for instance adapts the learning rate of a base optimizers like SGD using a meta-gradient descent procedure, although this does not remove the learning rate completely but replaces it with a less sensitive hypergradient step size. Mechanic (Cutkosky, Defazio, and Mehta 2023) pursues the same goal using a meta *online convex optimization* (OCO) algorithm.

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

$$\eta^* = \frac{||\theta^* - \theta_0||}{\sqrt{\sum_{t=0}^{T} ||g_t||^2}}. \tag{6}$$

Multiple recently introduced parameter free optimizers, have made use of this result. As its name suggests, the *Distance over Gradients* (DoG) (Ivgi, Hinder, and Carmon 2023) algorithm uses the maximum distance between the initial parameters and the parameters of all previous iterations as a surrogate for the unknown distance between initial and optimal parameters $||\theta^* - \theta_0||$ to compute its learning rate as

$$\eta^* = \frac{\max_{i<t} ||\theta_0 - \theta_i||}{\sqrt{\sum_{t=0}^{T} ||g_t||^2}}. \tag{7}$$

D-Adaptation by Defazio and Mishchenko (2023) on the other hand employs weighted dual averaging (Duchi, Agarwal, and Wainwright 2012) to calculate bounds on the distance between initial and optimal parameters, often denoted as D and use them to adapt the learning rate of a base optimization algorithm.

Despite the fact that parameter-free stochastic optimization techniques are inherently well-suited for the highly non-stationary streaming data (Schaul, Zhang, and LeCun 2013) and in some cases even developed based on online convex optimization, their application on data streams has rarely been investigated. We therefore investigate the suitability of some of the most prominent adaptive optimizers, listed in Table 1 for stream-based learning (ii).

There are also several lesser-known studies that have explored adaptive learning rates in specific application domains of online learning such as time series prediction (Miyaguchi and Kajino 2019; Fekri et al. 2021; Zhang 2021), federated learning (Canonaco et al. 2021), and recommender systems (Ferreira Jose, Enembreck, and Paul Barddal 2020). However, since we focus on general data stream applications in this paper, we did not investigate these techniques further.

is valid?

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 [1]	$\mathcal{O}(2D)$	$\mathcal{O}(1D)$	✗	✗
Mechanic	$\mathcal{O}(10D)$	$\mathcal{O}(1D)$	✓	✓
DoG [1]	$\mathcal{O}(5D)$	$\mathcal{O}(1D)$	✗	✓
DAdapt [1]	$\mathcal{O}(6D)$	$\mathcal{O}(2D)$	✗	✓

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 and based on pseudo-codes provided in the original works. Due to lack of comparability, we do not list provided convergence guarantees. Note that this is not a comprehensive list but an overview of some of the most prominent algorithms.

3 Differences between Batch and Online Learning

In a batch learning setting, optimizing the learning rate comes down to finding values that minimize the expected loss on a hold-out set of validation data at the end of the training process. Formally, we can denote this task as

$$\min_{\eta_0,\ldots,\eta_T} \mathbb{E}_{x\sim p_v(x)}[\mathcal{L}(x, \theta_T)], \tag{8}$$

where p_v is a distribution of held-out validation data and θ_T the parameters at the end of training, which for basic SGD are given by

$$\theta_T = \sum_{t=0}^{T} \eta_t \cdot g_t. \tag{9}$$

In online learning where data is generated incrementally, this notion of learning rate optimization is infeasible. Due to requirements **RQ1-RQ5** models operating in an online streaming environment must be evaluated in a *prequential* manner (Bifet et al. 2010), where each instance x_t in the data stream is first used to test and then to train the model ensuring that testing is done exclusively on unseen data.

Training in such a scenario can therefore be more accurately modeled as an *online convex optimization* (OCO) problem (Shalev-Shwartz 2011; Hazan 2016), where the optimizer suffers a loss $\mathcal{L}_t(\theta_t) = \mathcal{L}(x_t, \theta_t)$ and produces updated parameters θ_{t+1} at each iteration of the data stream.

The task of finding an optimal learning rate schedule in this setting, can be formulated as

$$\min_{\eta_0,\ldots,\eta_T} \sum_{t=0}^{T} \mathcal{L}(x_t, \theta_t), \tag{10}$$

where data samples x_t are drawn from a distribution $p_t(x)$.

Compared to Problem (8), Problem (10) features some key differences. Due to Requirement 1, the goal is to minimize the total sum of losses incurred over all timesteps

[1] Variant with SGD as the base algorithm.

of the prequential evaluation process, instead of minimizing the only the validation loss for the final parameters θ_T. This means that not only the loss achieved by the final parameters θ_T, but the loss suffered at every timestep of the stream contributes equally to the objective. Therefore, speed of convergence is of larger importance in the streaming setting, whereas the performance of the final parameters θ_T has relatively little impact. Since memory is limited (Requirement 1), it is also not possible to continue training on previously observed data as long as $\mathcal{L}$ decreases, which puts an even greater emphasis on quick adaptation. At the same time, a larger learning rate causing temporary loss increases, due to skipping over local minima can be suboptimal with respect to Problem (10) even if it eventually yields a lower loss.

Another difference to conventional batch learning is that the distribution p_t is time dependent, due to the fact that data streams might, and in practice most likely will, be subjected to change in the form of *concept drift*[1] over time. Under such circumstances, the optimal parameter values θ^* move throughout the progression of the stream requiring the model parameters to adapt.

3.1 Learning Rate Tuning

Concept drift also complicates the tuning of η, since even if data is available beforehand drift would eventually cause the stream to diverge from the distribution of data used for tuning. This effect, combined with the previously described differences in the evaluation scheme can cause conventional learning rate tuning to produce unsuitable results for stream-based learning. We therefore propose a slightly different online learning specific tuning approach approximating Problem (10), which we call *learning rate pre-tuning* in the following.

To emulate the targeted data stream we continually draw samples with replacement from the tuning data in a bootstrapping procedure instead of training on all data for multiple epochs. By doing so we aim to increase data variability, and therefore the resemblance to an actual data stream with random distributional shifts. We then optimize η with respect to the mean prequential performance over the emulated stream instead of the performance on a validation set. For this purpose we use a basic grid-search as is customary in batch learning (Defazio and Mishchenko 2023). We provide a detailed experimental evaluation of our approach in Section 4.

3.2 Learning Rate Adaptation

As described earlier, model parameters in streaming environments must be regularly adjusted to account for concept drift. To enhance the model's ability to do so, it appears intuitive, to increase the learning rate whenever distributional change occurs.

Based on this notion, Kuncheva and Plumpton (2008) introduced an adaptive schedule that uses the predictive losses as an indicator for concept drift. Their approach updates the learning rate using

$$\eta_{t+1} = \eta_t^{1+\bar{\mathcal{L}}_{t-M}-\bar{\mathcal{L}}_t}, \tag{11}$$

where $\bar{\mathcal{L}}_t$ is the running mean of M previous losses. By doing so, the authors aim to achieve higher stability, when data is stationary and losses decline and higher adaptability, when data is drifting and losses rise. While this approach seems intuitively sound, for an initial learning rate $\eta_0 \leq 1$ it bears a high risk of increasing up to a value of 1, since increases in loss caused by an excessive learning rate would lead to a feedback loop. Furthermore, loss plateaus that could be avoided by lowering η would instead cause η to remain stable, diminishing performance.

To offer the same potential benefits as Kuncheva and Plumpton (2008) approach while addressing its fundamental issues, we propose a simple adaptation to decaying learning rate schedules that resets η to its original value if a concept drift has been detected. An exponential schedule modified with our approach therefore yield learning rates

$$\eta_t = \eta_0 \cdot \gamma^{t-t_d}, \tag{12}$$

where t_d marks the timestep in which drift was last detected. As a result, feedback-loops are avoided assuming η_0 is small enough to not cause divergence and η_t can also decay in the presence of loss plateaus.

For the purpose of drift detection we apply ADWIN (Bifet and Gavaldà 2007) to the prequential losses. To avoid mistakenly detecting drops in loss as concept drifts, we use a one-tailed ADWIN variant that tests only for increases.

Our approach is similar to some *forgetting mechanisms* (Gama et al. 2014) commonly employed in conventional non-deep online learning, which improve model plasticity by partly (Bifet and Gavaldà 2009) or resetting the current model's parameters to their initial values. However, we hypothesize that this approach is not well suited for deep learning purposes. The reason for this is that, under the assumption of convexity, it requires that the newly initiated parameters be closer to the optimal parameters θ^* than the current parameters to be beneficial. For all but the most severe drifts, this seems highly unlikely. Nevertheless, we experimentally compare our approach with this mechanism in Section 4.

4 Experiments

To evaluate our hypotheses we performed computational experiments using the following setup[2]:

We used both synthetic and publicly available real-world classification datasets, with different sizes and types of concept drift, listed in Table 2.

With the purpose of generating similar datasets with different types of concept drift, we generated Random Radial Basis Function (RBF) datasets using the online learning framework River (Montiel et al. 2021). We then caused concept drift by either incrementally moving all RBF centroids

[1] We use concept drift as an umbrella term for any form of distributional shift.

[2] Our code is available at anonymous.4open.science/r/LODL-D458/.

or abruptly switching the random seed of the generator in the center of the stream.

We further employed the Electricity and Covertype (Blackard 1998) datasets, which are commonly used to evaluate online learning models, as well as the Insects datasets (Souza et al. 2020) with known types of concept drift. Covertype is accessible through the OpenML Platform (Vanschoren et al. 2014), while the remaining datasets are part of River.

As the model architecture, we used a single hidden layer MLP implemented in PyTorch (Paszke et al. 2019). To account for the different dimensionality of the selected data streams, with the number of hidden units equal to number of input features. This choice was informed by our experience that smaller models exhibit faster convergence and are therefore usually preferable over larger ones in online learning scenarios.

For selecting the decay factor γ in decaying learning rate schedules, we ran a grid search with three geometrically spaced values. We then used the value yielding the best accuracy for most datasets and the next smaller value for schedules with our resetting mechanism. To ensure a minimal level of adaptability even for longer streams, we set a lower bound at 10% of the base learning rate. Unless stated otherwise, we tuned the base learning rate η_0 of all but the parameter-free approaches using a grid search of ten geometrically spaced values. For the proposed learning rate resetting mechanism, we set the confidence level δ of ADWIN to 10^{-4}. For more details on our hyperparameter setup, refer to Appendix A.

For our evaluations we processed each dataset sequentially, emulating streams of mini-batches with 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. Since the prequential binary crossentropy used for training, can contain large outliers, we focus on the classification accuracy as a performance metric.

Type	Data Stream	Instances	Features	Classes
Synth.	RBF abrupt	20000	20	5
	RBF incremental	20000	20	5
Real	Insects abrupt	52848	33	6
	Insects incremental	57018	33	6
	Insects gradual	24150	33	6
	Covertype[3]	100000	54	7
	Electricity	45312	8	2

Table 2: Datasets used for experimental evaluations.

4.1 Drift Adaptation

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 the adaptation algorithm by Kuncheva and Plumpton (2008) (see Equation (3.2)) and complete model resetting, commonly used in online learning.

As can be seen in Table 3, our approach clearly outperforms both other drift adaptation techniques by a wide margin on all but Covertype, where weight resetting yielded slightly higher accuracy. It also compares favorably against static exponential decay on the RBF incremental, Covertype and Electricity datasets. However, aside from the results on RBF incremental the accuracy increases are negligible. Furthermore, learning rate resetting did not yield improvements for the Insects datasets regardless of the type of drift. This is also reflected in Figure 1, where the standard exponential schedule's accuracy initially rises faster and also recovers faster after the first concept drift, which is likely caused by its larger step size in the first half of the stream. Although the resetting mechanism was frequently triggered when no concept drift occurred, our results for the oracle resetting approach that was triggered only for timesteps with drift show that 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 for assuring adequate adaptability to concept drift throughout most data streams while granting better stability at later stages.

Overall, a slower static decay and a larger initial learning rate seems to be preferable over a more aggressive schedule with our drift resetting mechanism, unless severe concept drift as in RBF incremental can be expected.

With accuracy values within the standard deviation range of one another on all evaluated streams, step-wise decay displays almost identical performance to exponential decay. The cyclic schedule's accuracy for RBF incremental and Covertype on the other hand significantly exceeded that of the other static and adaptive schedules but lacks behind on all other streams. We also did not find improvements in convergence speed by an order of magnitude as observed by (Smith and Topin 2018) for the investigated scenario. Based on our results, the usefulness of cyclic learning rates for online learning applications therefore seems to be more data dependent than conventional decaying schedules.

4.2 Adaptive Learning Rates

To make inferences about the usefulness of different adaptive first-order optimization techniques for online deep learning, we performed prequential evaluations for the methods listed in Table 1.

From the results displayed in Table 4, it can be deduced that none of the approaches that require tuning a step size parameter stands out as generally superior for the investigated datasets. Rather, each of SGD, AdaGrad, and Adam achieve the best accuracy on two of the seven datasets. This once again underlines the data dependency of the optimizer performance. 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 play a significant role.

Due to its simplicity and favorable computational efficiency, it appears that standard SGD should be selected out of the non-parameter-free approaches if the characteristics of the targeted data stream are unknown. The SGD variant of Hypergradient Descent (Baydin et al. 2018) and WN-

[2]We used the first 100k from a total of 581k examples only.

	Schedule	RBF abrupt	RBF incr.	Covertype	Electricity	Insects abrupt	Insects gradual	Insects incr.
Static	Fixed	94.79±.32	70.95±2.89	83.42±.50	73.77±.40	71.50±.08	75.31±.21	60.48±.20
	Step	**94.87±.28**	70.19±3.02	82.89±.37	73.62±.53	**72.23±.27**	75.83±.21	61.18±.11
	Cyclic	94.79±.32	**74.96±.86**	**83.44±.08**	68.38±.81	71.74±.39	75.64±.06	60.48±.20
	Exponential	94.85±.29	70.23±2.40	82.95±.26	73.51±.48	72.19±.37	**75.91±.14**	**61.28±.16**
Adaptive	Weight Reset	69.96±.38	65.13±.80	83.12±.13	70.08±1.66	51.52±.90	62.55±2.34	34.11±.44
	Kuncheva	70.60±6.24	42.37±1.31	76.98±.15	67.06±.01	67.45±.50	72.43±.61	54.17±.30
	LR Reset (Ours)	94.83±.26	73.38±2.32	82.99±.20	**73.79±.62**	71.73±.20	75.52±.12	60.77±.08
	LR Reset Oracle	95.12±.21	—	—	—	71.88±.26	—	—

Table 3: Average prequential accuracy [%] for static (top) and drift adaptive learning rate schedules (bottom) with SGD. For LR Reset Oracle we manually reset the learning rate at timesteps where concept drift occurs.

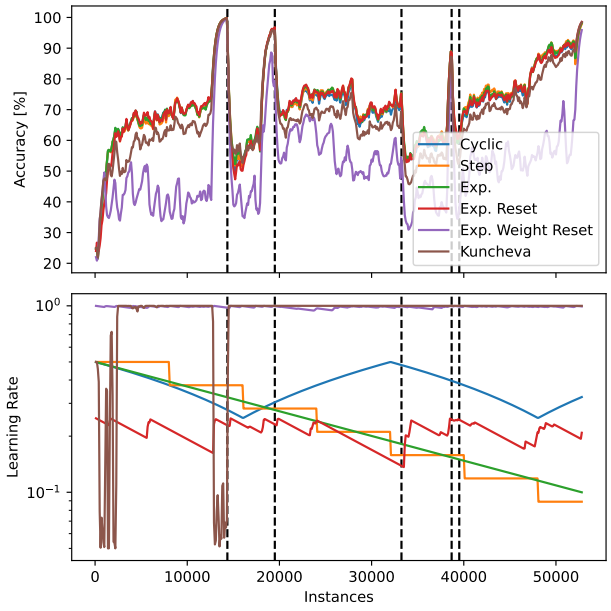

Figure 1: Prequential accuracy and learning rate for our drift resetting approach (see Equation (3.2)), drift adaptation by Kuncheva and Plumpton (2008) and different basic schedules on Insects abrupt. Concept drifts are marked by dashed lines. The accuracy is exponentially smoothed with a decay factor of 0.75.

Grad (Wu, Ward, and Bottou 2020) on the other hand seem to rarely be optimal choices.

In the category of learning rate free optimizers COCOB (Orabona and Tommasi 2017), outperformed its competitors on all but two datasets. Its accuracy also comes close to that or even exceeds that of the best tuned approaches. As the second best parameter-free method, DoG also comes within reach of the accuracy of the tuned methods, while offering much better runtime and memory efficiency compared to COCOB (see Table 1). WNGrad (Wu, Ward, and Bottou 2020) as well as the Mechanist (Cutkosky, Defazio, and Mehta 2023) and DAdaptation (Defazio and Mishchenko 2023) SGD variants on the other hand performed significantly worse than all other optimization approaches.

To gain additional insights into the investigated adaptive optimizers, we calculated their effective learning rates as $\frac{||\eta_t||}{\sqrt{D}}$, where $\eta_t \in \mathbb{R}^D$ is the vector of parameter specific learning rates. The resulting learning rate curves shown in Figure 1, provide an indication regarding the reason for the poor performance of WNGrad and DAdaption. Whereas the learning rate of DoG quickly approaches the tuned SGD learning rate, the two parameter-free methods diverge considerably from it. A possible cause for this could be the higher level of gradient noise introduced by the small mini-batches and concept drift associated with the online learning setting.

Another interesting observation that can be made in Figure 2 is that the learning rate of the best performing Adam features spikes for most change points, suggesting some form of adaptability to drift. Since the much worse performing Mechanic shows similar spikes, this is however unlikely to be a significant contributing factor to Adams high accuracy on Insects abrupt.

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 AdaGrad-like scaling, which creates larger learning rates for parameters with small and consistent gradients (Cutkosky, Defazio, and Mehta 2023). Therefore, the parameter updates generated by these approaches are not necessarily larger.

4.3 Learning Rate Tuning

We evaluated our proposed learning rate pre-tuning approach for MLPs with either one or three hidden layers and 64 or 128 hidden units per layer. To this end, we performed prequential evaluation runs with multiple values for the initial learning rate η_0 as well as the exponential decay factor γ, bootstrapping samples from a subset of 500 or 1000 instances from the beginning of each data stream, held out from the remaining data. At each iteration of this process we determine the hyperparameters with the best mean prequential accuracy. We use accuracy instead of loss as the selection criterion, due to the fact that the binary crossentropy employed for training commonly decreases by orders of magnitude throughout optimization. A selection based on the prequential loss would therefore only consider the initial

	Optimizer	RBF abrupt	RBF incr.	Covertype	Electricity	Insects abrupt	Insects gradual	Insects incr.
Tuned	SGD	**94.79±.32**	70.95±2.89	**83.42±.50**	73.77±.40	71.50±.08	75.31±.21	60.48±.20
	Adam	93.45±.30	69.26±5.14	79.01±.27	69.79±.54	**75.38±.24**	75.78±.74	**64.17±.13**
	AdaGrad	92.45±1.37	52.87±6.62	81.68±.35	**76.99±1.20**	74.87±.40	**77.15±.27**	62.51±.59
	WNGrad	87.30±.68	44.92±.73	76.98±.15	70.80±.59	66.25±.19	66.75±.40	56.14±.21
	HD	93.92±.31	**72.29±2.90**	83.36±.25	73.83±.32	70.67±.06	73.37±.21	59.92±.18
LR-Free	COCOB	**93.40±.38**	63.52±2.70	82.27±.46	**84.30±.56**	**74.75±.11**	**77.00±.05**	**63.65±.16**
	DoG	92.72±.59	**73.17±2.72**	**83.07±.64**	71.53±.70	70.59±.26	74.01±.21	59.66±.22
	DAdapt	74.91±4.22	45.47±2.75	76.69±.79	66.03±1.75	50.05±11.26	48.21±10.62	36.00±11.81
	Mechanic	88.94±.58	49.26±1.44	78.67±.18	50.73±7.60	55.31±21.47	65.80±.53	47.89±17.46

Table 4: Average prequential accuracy [%] for different adaptive optimizers and SGD.

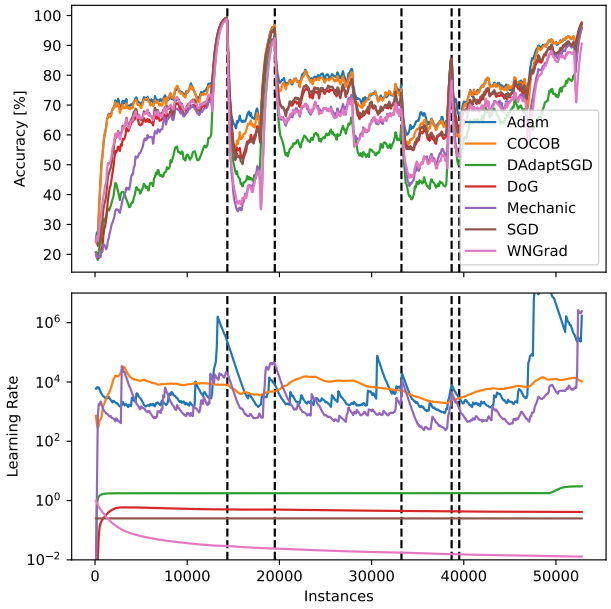

Figure 2: Prequential accuracy and learning rate for adaptive optimizers on Insects abrupt. Concept drifts are marked by dashed lines. Accuracy is exponentially smoothed with a decay factor of 0.75.

iterations of the tuning process.

Figure 3 shows the learning rate resulting from the pre-tuning process at each tuning step averaged over all real-world datasets. The bottom row of the figure displays the mean accuracy achieved when using this learning rate on the remaining data stream not used for tuning. Since we find our tuning procedure to be robust to the model size (see Appendix B) we focus on the results for the smallest network.

It can be seen that after overshooting initially, the tuned learning rate converges in the vicinity of optimal learning rate after 1000 iterations. While tuning with 1000 samples yields a better approximation of the optimal value, both subset sizes on average achieve significantly better learning rates for more than 1000 pre-tuning steps, than conventional tuning performed with 800 training and 200 validation samples. This is also clearly reflected in the accuracy scores, which already exceed conventional tuning after 500 iterations. Although our approach maintains a larger deviation from the optimal learning rate, it notably surpasses DoG in terms of accuracy. This advantage seemingly stems from the fact that excessively low learning rates have a more negative impact than excessively high ones for the evaluated scenario.

Figure 4 shows that when provided with sufficient data, pre-tuning also significantly outperforms DoG for a fixed learning rate. However, its performance benefit compared to DoG is considerably larger with a decaying schedule.

In conclusion, our proposed tuning approach enables significantly better learning rate selection for prequential evaluation on data streams compared to both conventional tuning as well as 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. Although, if computational efficiency is insignificant, the highly performant but expensive COCOB (Orabona and Tommasi 2017) or the slightly less performant and much less expensive DoG (Ivgi, Hinder, and Carmon 2023) may be more appropriate.

5 Conclusion

In this work, we investigate the influence and selection of the learning rate and optimization procedure with respect to deep learning 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 adapting 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 (Orabona and Tommasi 2017) and DoG (Ivgi, Hinder, and Carmon 2023) 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

Baydin, A. G.; Cornish, R.; Rubio, D. M.; Schmidt, M.; and Wood, F. 2018. Online Learning Rate Adaptation with Hy-

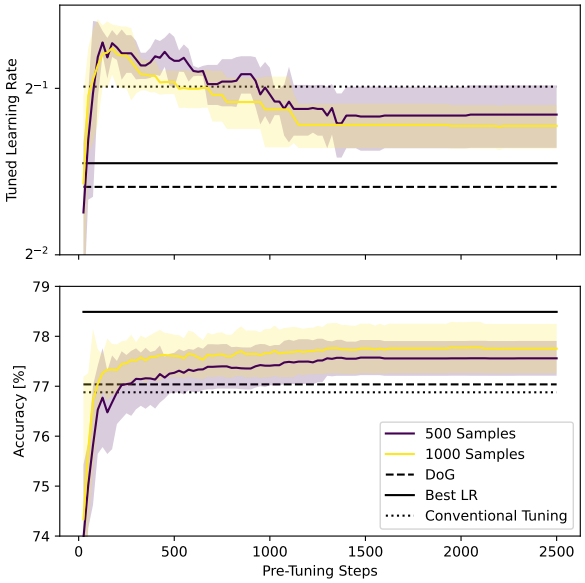

Figure 3: Pre-tuned LR (LR that maximizes accuracy on pre-tuning data) and resulting accuracy on data streams when using SGD and an exponential learning rate schedule with 500 or 1000 separate tuning samples. Results are averaged over all real-world datasets. The shaded area represents the 1σ-interval.

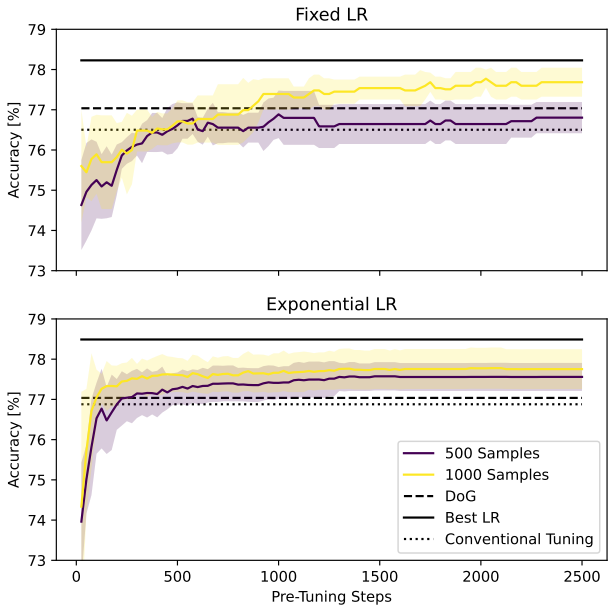

Figure 4: Accuracy achieved by pre-tuning on 500 or 1000 samples when using SGD with a fixed LR schedule (top) or an exponential schedule (bottom), averaged over all real-world datasets. The shaded area represents the 1σ-interval.

pergradient Descent. In *ICLR Proceedings*.

Bengio, Y. 2012. Practical Recommendations for Gradient-Based Training of Deep Architectures. arxiv:1206.5533.

Bifet, A.; and Gavaldà, R. 2007. Learning from Time-Changing Data with Adaptive Windowing. In *Proceedings of the 2007 SIAM International Conference on Data Mining*, 443–448. Society for Industrial and Applied Mathematics. ISBN 978-0-89871-630-6 978-1-61197-277-1.

Bifet, A.; and Gavaldà, R. 2009. Adaptive Learning from Evolving Data Streams. In Adams, N. M.; Robardet, C.; Siebes, A.; and Boulicaut, J.-F., eds., *Advances in Intelligent Data Analysis VIII*, Lecture Notes in Computer Science, 249–260. Berlin, Heidelberg: Springer. ISBN 978-3-642-03915-7.

Bifet, A.; Holmes, G.; Kirkby, R.; and Pfahringer, B. 2010. MOA: Massive Online Analysis. *Journal of Machine Learning Research*, 11.

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

Canonaco, G.; Bergamasco, A.; Mongelluzzo, A.; and Roveri, M. 2021. Adaptive Federated Learning in Presence of Concept Drift. In *2021 International Joint Conference on Neural Networks (IJCNN)*, 1–7.

Chaudhari, P.; Choromanska, A.; Soatto, S.; LeCun, Y.; Baldassi, C.; Borgs, C.; Chayes, J.; Sagun, L.; and Zecchina, R. 2017. Entropy-SGD: Biasing Gradient Descent Into Wide Valleys. arxiv:1611.01838.

Cutkosky, A.; Defazio, A.; and Mehta, H. 2023. Mechanic: A Learning Rate Tuner. arxiv:2306.00144.

Defazio, A.; and Mishchenko, K. 2023. Learning-Rate-Free Learning by D-Adaptation. arxiv:2301.07733.

Duchi, J. C.; Agarwal, A.; and Wainwright, M. J. 2012. Dual Averaging for Distributed Optimization: Convergence Analysis and Network Scaling. *IEEE Transactions on Automatic Control*, 57(3): 592–606.

Duchi, J. C.; Hazan, E.; and Singer, Y. 2011. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *Journal of Machine Learning Research*, 12(61): 2121–2159.

Fekri, M. N.; Patel, H.; Grolinger, K.; and Sharma, V. 2021. Deep Learning for Load Forecasting with Smart Meter Data: Online Adaptive Recurrent Neural Network. *Applied Energy*, 282: 116177.

Ferreira Jose, E.; Enembreck, F.; and Paul Barddal, J. 2020. ADADRIFT: An Adaptive Learning Technique for Long-history Stream-based Recommender Systems. In *2020 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, 2593–2600. Toronto, ON, Canada: IEEE. ISBN 978-1-72818-526-2.

Gama, J.; Žliobaitė, I.; Bifet, A.; Pechenizkiy, M.; and Bouchachia, A. 2014. A Survey on Concept Drift Adaptation. *ACM Computing Surveys*, 46(4): 1–37.

Hazan, E. 2016. Introduction to Online Convex Optimization. *Foundations and Trends® in Optimization*, 2(3-4): 157–325.

Hochreiter, S.; and Schmidhuber, J. 1997. Flat Minima. *Neural Computation*, 9(1): 1–42.

Ivgi, M.; Hinder, O.; and Carmon, Y. 2023. DoG Is SGD's Best Friend: A Parameter-Free Dynamic Step Size Schedule. arxiv:2302.12022.

Kingma, D. P.; and Ba, J. 2017. Adam: A Method for Stochastic Optimization. arxiv:1412.6980.

Kuncheva, L. I.; and Plumpton, C. O. 2008. Adaptive Learning Rate for Online Linear Discriminant Classifiers. In Da Vitoria Lobo, N.; Kasparis, T.; Roli, F.; Kwok, J. T.; Georgiopoulos, M.; Anagnostopoulos, G. C.; and Loog, M., eds., *Structural, Syntactic, and Statistical Pattern Recognition*, volume 5342, 510–519. Berlin, Heidelberg: Springer Berlin Heidelberg. ISBN 978-3-540-89688-3 978-3-540-89689-0.

Miyaguchi, K.; and Kajino, H. 2019. Cogra: Concept-Drift-Aware Stochastic Gradient Descent for Time-Series Forecasting. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33(01): 4594–4601.

Montiel, J.; Halford, M.; Mastelini, S. M.; Bolmier, G.; Sourty, R.; Vaysse, R.; Zouitine, A.; Gomes, H. M.; Read, J.; Abdessalem, T.; et al. 2021. River: Machine Learning for Streaming Data in Python.

Orabona, F.; and Tommasi, T. 2017. Training Deep Networks without Learning Rates Through Coin Betting. In *NIPS*.

Paszke, A.; Gross, S.; Massa, F.; Lerer, A.; Bradbury, J.; Chanan, G.; Killeen, T.; Lin, Z.; Gimelshein, N.; Antiga, L.; Desmaison, A.; Kopf, A.; Yang, E.; DeVito, Z.; Raison, M.; Tejani, A.; Chilamkurthy, S.; Steiner, B.; Fang, L.; Bai, J.; and Chintala, S. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In Wallach, H.; Larochelle, H.; Beygelzimer, A.; d'Alché-Buc, F.; Fox, E.; and Garnett, R., eds., *NeurIPS Proceedings*, 8024–8035. Curran Associates, Inc.

Schaul, T.; Zhang, S.; and LeCun, Y. 2013. No More Pesky Learning Rates. arxiv:1206.1106.

Shalev-Shwartz, S. 2011. Online Learning and Online Convex Optimization. *Foundations and Trends® in Machine Learning*, 4(2): 107–194.

Smith, L. N. 2017. Cyclical Learning Rates for Training Neural Networks. arxiv:1506.01186.

Smith, L. N.; and Topin, N. 2018. Super-Convergence: Very Fast Training of Neural Networks Using Large Learning Rates. arxiv:1708.07120.

Smith, S. L.; Kindermans, P.-J.; Ying, C.; and Le, Q. V. 2018. Don't Decay the Learning Rate, Increase the Batch Size. arxiv:1711.00489.

Smith, S. L.; and Le, Q. V. 2018. A Bayesian Perspective on Generalization and Stochastic Gradient Descent. arxiv:1710.06451.

Souza, V. M. A.; dos Reis, D. M.; Maletzke, A. G.; and Batista, G. E. A. P. A. 2020. Challenges in Benchmarking Stream Learning Algorithms with Real-world Data. *Data Mining and Knowledge Discovery*, 34(6): 1805–1858.

Sutskever, I.; Martens, J.; Dahl, G.; and Hinton, G. 2013. On the Importance of Initialization and Momentum in Deep Learning. In *Proceedings of the 30th International Conference on Machine Learning*, 1139–1147. PMLR.

Tieleman, T.; and Hinton, G. 2012. Lecture 6.5-Rmsprop: Divide the Gradient by a Running Average of Its Recent Magnitude. In *COURSERA: Neural Networks for Machine Learning*. Coursera.

van Erven, T.; and Koolen, W. M. 2016. MetaGrad: Multiple Learning Rates in Online Learning. arxiv:1604.08740.

Vanschoren, J.; van Rijn, J. N.; Bischl, B.; and Torgo, L. 2014. OpenML: Networked Science in Machine Learning. *ACM SIGKDD Explorations Newsletter*, 15(2): 49–60.

Wu, X.; Ward, R.; and Bottou, L. 2020. WNGrad: Learn the Learning Rate in Gradient Descent. arxiv:1803.02865.

Wu, Y.; Liu, L.; Bae, J.; Chow, K.-H.; Iyengar, A.; Pu, C.; Wei, W.; Yu, L.; and Zhang, Q. 2019. Demystifying Learning Rate Policies for High Accuracy Training of Deep Neural Networks. arxiv:1908.06477.

Zeiler, M. D. 2012. ADADELTA: An Adaptive Learning Rate Method. arxiv:1212.5701.

Zhang, W. 2021. POLA: Online Time Series Prediction by Adaptive Learning Rates. arxiv:2102.08907.

A Hyperparameter values

In our experiments we used the following hyperparameter settings.

Schedule	Static
Exponential	$\gamma = 1 - 2^{-13}$
Exp. Reset	$\gamma = 1 - 2^{-12}, \delta = 0.0001$
Step	$\gamma = 0.75, s = 2000$
Cyclic	$\hat{\eta} = 0.25, s = 8000$

Table 5: Learning Rate Schedule Hyperparameters.

Optimizer	Learning Rate
SGD	$\{2^1, 2^0, \ldots, 2^{-8}\}$
Adam	$\{2^{-3}, 2^{-4}, \ldots, 2^{-12}\}$
AdaGrad	$\{2^1, 2^0, \ldots, 2^{-8}\}$
WNGrad	$\{10^{1.25}, 10^{0.75}, \ldots, 10^{-7.75}\}$
SGD-HD	$\{2^{-3}, 2^{-4}, \ldots, 2^{-12}\}$
COCOB	100
DoG	1
DAdaptSGD	1
Mechanic	0.01

Table 6: Search spaces for learning rates of different optimizers.

B Model Sizes in Learning Rate Tuning

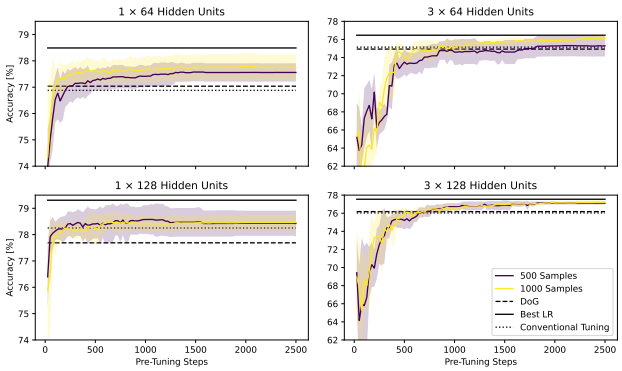

Figure 5: Accuracy achieved by pre-tuning on 500 or 1000 samples when using SGD with an exponential schedule on different network sizes, averaged over all real-world datasets. The shaded area represents the 1σ-interval.