Learning Rate Optimization in Online Deep Learning

Anonymous[1]

No Institute Given

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 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 has to adapt 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 deep learning.

Keywords: Data streams · Learning Rate · Neural Networks.

1 Introduction

Deep learning models have demonstrated 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. Nearly all modern deep learning applications, use first-order stochastic optimization methods like *stochastic gradient descent*, which iteratively update the parameters of the underlying model based on gradient information, for this purpose. 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). Based on prior knowledge, the learning rate can for instance be set as a fixed value or a schedule altering the step size over time. Alternatively, one could use an adaptive learning rate technique, which considers historical gradient information to modify the learning rate at each iteration.

In batch learning scenario where all training data is available a priori, the aforementioned methods are well researched. Despite the increasing prevalence of online learning environments, where data becomes available step by step as

part of a data stream, their use in such scenarios has however received little attention in research.

According to Bifet *et al.* [4] 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 introduce additional challenges when it comes to the selection of an appropriate optimizer and learning rate.

To enable more informed decisions when it comes to learning rate optimization, we provide insight into these challenges and empirically evaluate commonly used optimization techniques in an online learning setting (i). We further introduce a *drift reset* mechanism to adapt the learning rate to concept drifts, that commonly occur in streaming environments (ii). Lastly, we propose a *pre-tuning* approach for effectively optimizing the learning rate of online deep learning models *pre-stream* (see Figure 1) (iii).

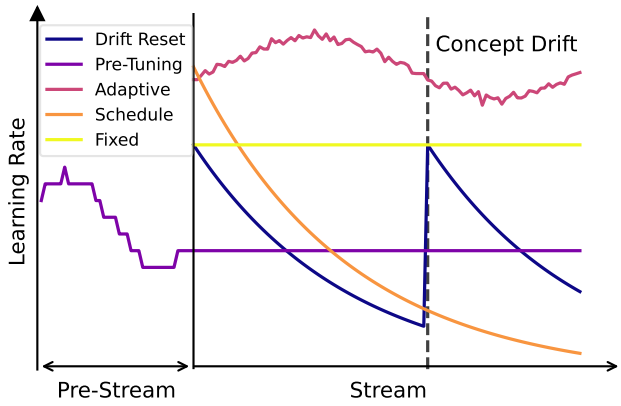


Fig. 1: Overview of different learning rate optimization approaches.

2 Learning Rate in First-Order Optimization

In the following, we will explain the theoretical background of first-order stochastic optimization enabling modern deep learning models.

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 $\theta_t = \theta_t - \eta_t \cdot g_t$, 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 [2]. In fact, Smith and Le [24], found that the "noise scale" of SGD is directly tied to η [24].

2.1 Learning Rate Schedules

Often times, the performance of a model can be improved by using a schedule that changes the learning rate as training progresses [30]. 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 [13, 6].

Commonly used forms of decay are exponential decay, where η_t calculates as $\eta_t = \eta_0 \cdot \gamma^t$, with $\gamma < 1$, and stepwise decay, which for a regular interval between steps of length s is given as $\eta_0 \cdot \gamma^{\lfloor t/s \rfloor}$. Another common approach involves decaying η every time the training loss plateaus for a set number of iterations.

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

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

with $\hat{\eta}$ being the learning rate at the middle of each cycle of length s. Some studies [22, 23] have found cyclic schedules to significantly speed up the convergence of neural networks even when compared to adaptive techniques like Adam in some cases [15]. While there are many alternatives, in this work we focus on exponential, stepped and cyclic learning rates, as some of the most commonly used generic schedules. For a comprehensive overview and detailed analysis on learning rate policies, refer to Wu *et al.* [30].

2.2 Adaptive Learning Rates

Various 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 [10]. One of the earlier techniques in this category is *AdaGrad* [10], 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 like AdaDelta [31] and RMSProp [26], subsequently built on AdaGrad's scaling approach. The same applies to the commonly used Adam optimizer [15], that additionally takes a momentum term of past gradients into account to speed up the convergence for parameters with consistent derivatives. Another optimizer building on AdaGrad is *WNGrad* [29], which adaptively scales each parameter update based on the squared sum of past gradients.

So called parameter-free variants of SGD aim to eliminate the learning rate entirely by optimizing it as training progresses. For example, the *COCOB* algorithm [18] models parameter optimization as a gambling problem, where the goal is to maximize the rewards from betting on each gradient. The resulting strategy corresponds to running a meta-optimization algorithm that estimates the expected optimal learning rate [18]. Several other contributions [27, 1, 7] have also used the idea of learning η via a meta-optimization process. The *hypergradient descent* (HD) approach [1] 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 [7] 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

$$\eta^* = \frac{||\theta_0 - \theta^*||}{\sqrt{\sum_{t=0}^{T} ||g_t||^2}}. \tag{3}$$

Multiple parameter-free optimizers, make use of this notion. As its name suggests, the *Distance over Gradients* (DoG) [14] algorithm estimates the unknown numerator in Equation 3 as the maximum distance $\max_{i<t} ||\theta_0 - \theta_i||$ between the initial parameters and the parameters of all previous iterations. DoG additionally makes use of polynomial decay averaging as proposed by Shamir and Zhang [21]. *D-Adaptation* by Defazio and Mishchenko [8] on the other hand employs weighted dual averaging [9] to calculate bounds on the distance between initial and optimal parameters. Although adaptive optimization techniques seem intuitively well-suited for non-stationary data, their application on 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.

[1] Variant with SGD as the base algorithm.

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

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 denote this task as

$$\min_{\eta_0,\ldots,\eta_T} \mathbb{E}_{x\sim p_v(x)}[\mathcal{L}(x,\theta_T)], \tag{4}$$

where p_v is a distribution of held-out validation data and θ_T the parameters at the end of training. In online learning where data is generated incrementally, this notion of learning rate optimization is infeasible. Due to requirements **R1-R5**, models operating in an online learning environment should be evaluated in a *prequential* manner [4], where each sample x_t in the data stream is first used to test and then to train the model ensuring testing is done exclusively on unseen data.

Training in such a scenario can therefore be more accurately modeled as an online convex optimization problem [20, 12], 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

$$\min_{\eta_0,\ldots,\eta_T} \sum_{t=0}^{T} \mathcal{L}_t(\theta_t). \tag{5}$$

Compared to Equation (4), Equation (5) features some key differences. Due to the requirement of being able to predict at any time (**R4**), the goal is to minimize the total sum of losses incurred over all timesteps of the prequential evaluation process, instead of minimizing only the validation loss for the final parameters θ_T. This means that the loss suffered at every timestep of the stream contributes equally to the objective. Therefore, the speed of convergence is more important in the streaming setting, while the performance of the final θ_T parameter 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 Equation (5) even if it eventually yields a lower loss.

Another difference to conventional batch learning is that the loss function $\mathcal{L}_t$ is time dependent, due to the fact that data streams are commonly subjected to distributional change in the form of *concept drift* 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

Tuning the learning rate of an online machine learning model is a challenging task owing to the possibility of concept drift that may cause data stream to move away from the distribution used for tuning the model. This effect, combined with the previously described differences in the evaluation scheme may cause conventional learning rate tuning to produce unsuitable results for stream-based learning. We therefore propose a modified tuning approach, approximating Equation (5), 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 [8]. We provide a detailed experimental evaluation of our approach in Section 4.

3.2 Learning Rate Adaptation

As previously noted, concept drift requires the model parameters to repeatedly adapt. When interpreting the post-drift training as a new online optimization problem, the worst-case optimal learning rate can be calculated according to Equation 3 substituting the initial parameter values θ_0 with the values at the time of drift onset θ_{t_d}. As a result, more severe drifts, causing θ^* to move away from θ_{t_d}, may benefit from larger learning rates.

Based on this notion, Kuncheva and Plumpton [16] introduced an adaptive schedule that uses the predictive losses as an indicator for concept drift. Their approach updates the learning rate using

$$\eta_{t+1} = \eta_t^{1+\bar{\mathcal{L}}_{t-M}-\bar{\mathcal{L}}_t}, \tag{6}$$

where $\bar{\mathcal{L}}_t$ is the running mean of M previous losses. By doing so, the authors aim to achieve higher stability, when losses decline and higher adaptability when losses rise. While this approach seems intuitively sound, for an initial learning rate $\eta_0 \leq 1$ it can cause an instable learning rate, 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 [16] 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

$$\eta_t = \eta_0 \cdot \gamma^{t-t_d}, \tag{7}$$

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 [3] 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* commonly employed in conventional non-deep online learning [11]. To improve model plasticity, such mechanisms partly or completely reset the current model 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, the newly initiated parameters must 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 perform computational experiments using the following setup[2]:

We use 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 generate *Random Radial Basis Function* (RBF) datasets using the online learning framework *River* [17]. We then caused concept drift by incrementally moving data centroids or abruptly switching the random seed of the generator. We further employ the *Electricity* and *Covertype* [5] datasets, which are commonly used to evaluate online learning models, as well as the Insects datasets [25] with known types of concept drift. *Covertype* is accessible through the *OpenML* Platform [28], while the remaining datasets are part of River.

As the model architecture, we use a single hidden layer MLP implemented in *PyTorch* [19]. To account for the different dimensionality of the selected data streams, with the number of hidden units equal to number of input features. This

[2] Code available at `anonymous.4open.science/r/LODL-D458/`.

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

Table 2: Datasets used for experimental evaluations.

choice is based on our experience that smaller models exhibit faster convergence and are therefore usually the most suitable in online learning scenarios.

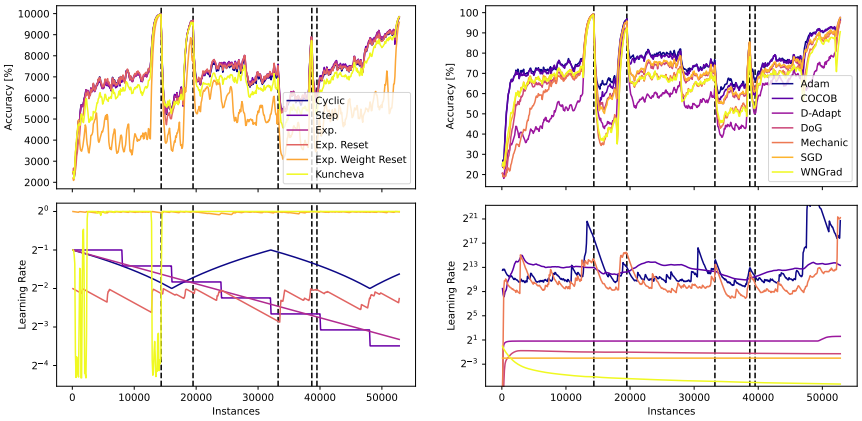
We tune the base learning rate η_0 of all but the parameter-free approaches using a grid search of ten geometrically spaced values. To ensure a minimal level of adaptability, we set a lower bound at 10% of the base learning rate. Parameter-free algorithms are configured with the author's suggested parameter values and paired with SGD as a base optimizer in the case of HD, Mechanic and D-Adaptation. 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 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 occasionally produced large outliers, we focus on the classification accuracy as a performance metric.

4.1 Drift Adaptation

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 1σ interval of best values are underlined.

	Schedule	RBF abrupt	RBF incr.	Covertype	Electricity	Insects abrupt	Insects gradual
Static	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**
Adaptive	Weight Reset	69.96±.38	65.13±.80	83.12±.13	70.08±1.66	51.52±.90	62.55±2.34
	Kuncheva [16]	70.60±6.24	42.37±1.31	76.98±.15	67.06±.01	67.45±.50	72.43±.61
	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	—

[3] We use the first 100k from a total of 581k examples only.



(a) Schedules (b) Adaptive Optimizers

Fig. 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.

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 [16] (see Equation (3.2)) and model weight 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 2a, where the standard exponential schedule's accuracy initially rises faster and 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 frequently acted 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 a data stream while granting better stability at later stages. Overall, a slower but static decay paired with 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* is expected.

With accuracy values within the 1σ interval 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 [23] 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

Table 4: Average prequential accuracy [%] for adaptive optimizers and SGD. We used SGD as the base optimizer for HD, Mechanic and D-Adapt. Best values are shown in **bold**, values within 1σ interval of best values are 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 [15]	93.45±.30	69.26±5.14	79.01±.27	69.79±.54	**75.38±.24**	75.78±.74
	AdaGrad [10]	92.45±1.37	52.87±6.62	81.68±.35	**76.99±1.20**	74.87±.40	**77.15±.27**
	WNGrad [29]	87.30±.68	44.92±.73	76.98±.15	70.80±.59	66.25±.19	66.75±.40
	HD [1]	93.92±.31	**72.29±2.90**	83.36±.25	73.83±.32	70.67±.06	73.37±.21
LR-Free	COCOB [18]	**93.40±.38**	63.52±2.70	82.27±.46	**84.30±.56**	**74.75±.11**	**77.00±.05**
	DoG [14]	92.72±.59	**73.17±2.72**	**83.07±.64**	71.53±.70	70.59±.26	74.01±.21
	D-Adapt [8]	74.91±4.22	45.47±2.75	76.69±.79	66.03±1.75	50.05±11.26	48.21±10.62
	Mechanic [7]	88.94±.58	49.26±1.44	78.67±.18	50.73±7.60	55.31±21.47	65.80±.53

To judge the usefulness of different adaptive first-order optimization techniques for online learning, we ran prequential evaluations with all 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 be significant.

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 (HD) [1] and WNGrad [29] on the other hand seem to rarely be optimal choices.

In the category of learning rate free optimizers COCOB [18], 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 [7] and D-Adaptation [8], which we ran with SGD as the base optimizer performed significantly worse than their competitors on the evaluated streams.

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 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 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 data streams.

Another interesting observation in Figure 2b 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 Adam's high accuracy on *Insects abrupt*. Instead, it likely stems from its second moment scaling, which is also part 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 [7]. Therefore, the parameter updates generated by these approaches are not necessarily larger.

4.3 Learning Rate Tuning

We assessed our suggested approach of pre-tuning learning rates for MLPs with 64 or 128 hidden units per layer and either one or three hidden layers. In the following part, we focus on the smallest network that is the most representative for streaming applications. The results of all architectures are reported in Appendix **??**.

Prequential evaluation runs were performed with a range of initial learning rates η_0 and exponential decay factors γ. From each data stream, we select a subset of either 500 or 1000 instances at the beginning by holding them out from the remaining data. In the pre-tuning process, we then bootstrap samples from the held out data to emulate a stream and determine the learning rate yielding the best mean prequential accuracy at each step. We use accuracy instead of loss as the selection criterion, because 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 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.

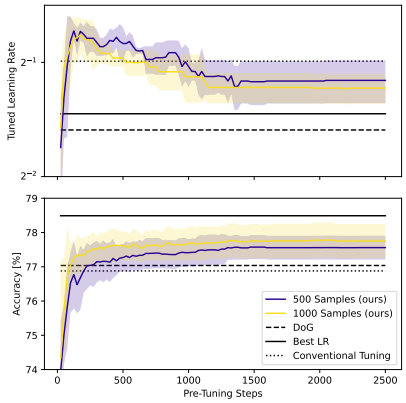


Fig. 3: Pre-tuned LR (LR that maximizes accuracy on pre-tuning data) and resulting accuracy on data streams when using SGD and an exponential learning rate schedule with 500 or 1000 separate tuning samples. Results are averaged over all real-world datasets. The shaded area represents the 1σ-interval.

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 reflected in the accuracy scores, which exceed conventional tuning after 500 iterations. Our approach achieves notably higher accuracy than DoG, which we selected as a baseline due to being best performing parameter-free optimizer with a global learning rate. This is the case despite deviating further from the optimal learning rate, likely due to excessively low learning rates having a larger impact than excessively high ones in the evaluated scenario.

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. Although, if computational efficiency is insignificant, the highly performant but expensive COCOB [18] or the slightly less performant and much less expensive DoG [14] 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 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

1. Baydin, A.G., Cornish, R., Rubio, D.M., Schmidt, M., Wood, F.: Online Learning Rate Adaptation with Hypergradient Descent. In: ICLR Proceedings (2018)
2. Bengio, Y.: Practical Recommendations for Gradient-Based Training of Deep Architectures, (2012).
3. Bifet, A., Gavaldà, R.: 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 (2007). https://doi.org/10.1137/1.9781611972771.42
4. Bifet, A., Holmes, G., Kirkby, R., Pfahringer, B.: MOA: Massive Online Analysis. Journal of Machine Learning Research **11** (2010)
5. Blackard, J.: Covertype, UCI Machine Learning Repository (1998).
6. Chaudhari, P. *et al.*: Entropy-SGD: Biasing Gradient Descent Into Wide Valleys, (2017).
7. Cutkosky, A., Defazio, A., Mehta, H.: Mechanic: A Learning Rate Tuner, (2023). http://arxiv.org/abs/2306.00144.
8. Defazio, A., Mishchenko, K.: Learning-Rate-Free Learning by D-Adaptation, (2023). http://arxiv.org/abs/2301.07733.
9. Duchi, J.C., Agarwal, A., Wainwright, M.J.: Dual Averaging for Distributed Optimization: Convergence Analysis and Network Scaling. IEEE Transactions on Automatic Control **57**(3), 592–606 (2012). https://doi.org/10.1109/TAC.2011.2161027
10. Duchi, J.C., Hazan, E., Singer, Y.: Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. Journal of Machine Learning Research **12**(61), 2121–2159 (2011)
11. Gama, J., Žliobaitė, I., Bifet, A., Pechenizkiy, M., Bouchachia, A.: A Survey on Concept Drift Adaptation. ACM Computing Surveys **46**(4), 1–37 (2014). https://doi.org/10.1145/2523813
12. Hazan, E.: Introduction to Online Convex Optimization. Foundations and Trends® in Optimization **2**(3-4), 157–325 (2016). https://doi.org/10.1561/2400000013
13. Hochreiter, S., Schmidhuber, J.: Flat Minima. Neural Computation **9**(1), 1–42 (1997). https://doi.org/10.1162/neco.1997.9.1.1
14. Ivgi, M., Hinder, O., Carmon, Y.: DoG Is SGD's Best Friend: A Parameter-Free Dynamic Step Size Schedule, (2023). http://arxiv.org/abs/2302.12022.
15. Kingma, D.P., Ba, J.: Adam: A Method for Stochastic Optimization, (2017).
16. Kuncheva, L.I., Plumpton, C.O.: Adaptive Learning Rate for Online Linear Discriminant Classifiers. In: Structural, Syntactic, and Statistical Pattern Recognition. Ed. by N. Da Vitoria Lobo *et al.*, pp. 510–519. Springer Berlin Heidelberg, Berlin, Heidelberg (2008). https://doi.org/10.1007/978-3-540-89689-0_55

17. Montiel, J. *et al.*: River: Machine Learning for Streaming Data in Python. (2021)
18. Orabona, F., Tommasi, T.: Training Deep Networks without Learning Rates Through Coin Betting. In: NIPS (2017)
19. Paszke, A. *et al.*: PyTorch: An Imperative Style, High-Performance Deep Learning Library. In: Wallach, H. (ed.) NeurIPS Proceedings, pp. 8024–8035. Curran Associates, Inc. (2019). http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf
20. Shalev-Shwartz, S.: Online Learning and Online Convex Optimization. Foundations and Trends® in Machine Learning **4**(2), 107–194 (2011). https://doi.org/10.1561/2200000018
21. Shamir, O., Zhang, T.: Stochastic Gradient Descent for Non-smooth Optimization: Convergence Results and Optimal Averaging Schemes, (2012). http://arxiv.org/abs/1212.1824.
22. Smith, L.N.: Cyclical Learning Rates for Training Neural Networks, (2017). https://doi.org/10.48550/arXiv.1506.01186.
23. Smith, L.N., Topin, N.: Super-Convergence: Very Fast Training of Neural Networks Using Large Learning Rates, (2018). http://arxiv.org/abs/1708.07120.
24. Smith, S.L., Le, Q.V.: A Bayesian Perspective on Generalization and Stochastic Gradient Descent, (2018). https://doi.org/10.48550/arXiv.1710.06451.
25. Souza, V.M.A., dos Reis, D.M., Maletzke, A.G., Batista, G.E.A.P.A.: Challenges in Benchmarking Stream Learning Algorithms with Real-world Data. Data Mining and Knowledge Discovery **34**(6), 1805–1858 (2020). https://doi.org/10.1007/s10618-020-00698-5
26. Tieleman, T., Hinton, G.: Lecture 6.5-Rmsprop: Divide the Gradient by a Running Average of Its Recent Magnitude. In: COURSERA: Neural Networks for Machine Learning. Coursera (2012)
27. van Erven, T., Koolen, W.M.: MetaGrad: Multiple Learning Rates in Online Learning, (2016). https://doi.org/10.48550/arXiv.1604.08740.
28. Vanschoren, J., van Rijn, J.N., Bischl, B., Torgo, L.: OpenML: Networked Science in Machine Learning. ACM SIGKDD Explorations Newsletter **15**(2), 49–60 (2014). https://doi.org/10.1145/2641190.2641198
29. Wu, X., Ward, R., Bottou, L.: WNGrad: Learn the Learning Rate in Gradient Descent, (2020).
30. Wu, Y. *et al.*: Demystifying Learning Rate Policies for High Accuracy Training of Deep Neural Networks, (2019). http://arxiv.org/abs/1908.06477.
31. Zeiler, M.D.: ADADELTA: An Adaptive Learning Rate Method, (2012).