Optimizing the Learning Rate for Online Training of Neural Networks

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, and many more, 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 distributional shifts, also known as *concept drift*, pose additional challenges. Therefore, the findings on batch learning may not apply 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 essential 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. The above methods are well-researched in batch learning scenarios, where all training data is available a priori. 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.

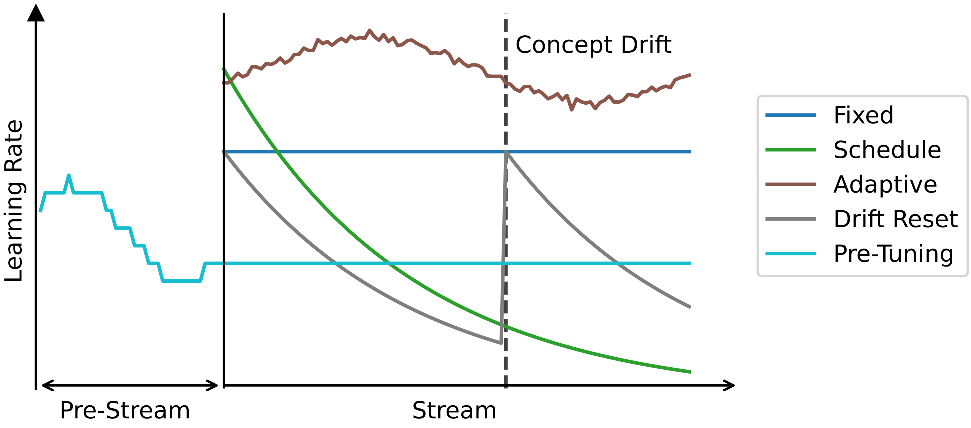


Figure 1: Overview of different learning rate optimization approaches.

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 data that may be available prior to stream-based learning (*pre-stream*) for 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 $\boldsymbol{x}_t$, 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 concerning η is that increasing it speeds up convergence but 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).

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

[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.

learning rates that let η oscillate between two values over a predefined interval. For a triangular cycle, the learning rate is defined 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, 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 step (Duchi et al., 2011).

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 completely 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 (Schraudolph, 1999; 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. However, 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, while *stochastic meta-descent* (SMD) by Schraudolph (1999), which extends Sutton's work on linear systems (Sutton, 1992), uses second-order information for local adaptations of the learning rate.

Research has shown that in an OCO 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 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.

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]	$\mathcal{O}(6d)$	$\mathcal{O}(2d)$	✗	✓

[2] Variant with SGD as the underlying optimizer.

Although adaptive optimization techniques seem intuitively well suited for non-stationary data, their application to data streams has yet to be 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

Optimizing the learning rate in a batch learning setting involves minimizing the expected loss on a hold-out validation data set. 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 critical 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 impossible to continue training on previously observed data as long as the loss decreases, putting 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 because 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, limited amounts of data may be available prior to the stream learning process in many real-world applications. Such data can then be used to tune the learning rate of the model before stream deployment (pre-stream). 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 in algorithm 1, any parameter search technique could be used for this purpose. We provide a detailed experimental evaluation of our approach in Section 4.

Algorithm 1 Pre-Stream Learning Rate Tuning for Online Learning Models

```
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 $n_{\text{steps}}$
$\mathbb{S} \leftarrow \{(\boldsymbol{x}_i, y_i) \sim \hat{p}_{\mathbb{X}} | \forall i \in \{1, \dots, n_{\text{steps}}\}\}$    ▷ Create artificial stream by sampling $\mathbb{X}$ with replacement.
$v^* \leftarrow -\infty$
for $\eta$ in $\mathbb{H}$, $\gamma$ 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 $\eta$.
    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 $\eta^*, \gamma^*$    ▷ 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 either *ADWIN* (Bifet & Gavaldà, 2007) or the Kolmogorov-Smirnov test (Massey, 1951) to the prequential losses. To avoid detecting loss decreases as concept drifts, we test only for increases of the prequential loss.

Our approach is similar to 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 unsuitable 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.~~ While the simplest implementation of forgetting resets all parameters, there is a variety of more targeted approaches for increasing model plasticity. For instance, Dohare et al. (2023) suggest computing a utility metric and resetting neurons that receive a low utility value, while Paik et al. (2019) instead adapt the learning rate for each neuron using a utility metric. Similarly, Elsayed & Mahmood (2023) limits changes to high-utility-weights while intentionally perturbing low-utility-weights.

~~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 synthetic and publicly available real-world classification datasets with different sizes and types of concept drift, listed in Table 2. Our evaluations include the *Agrawal* stream-generator (Agrawal et al., 1993) for which we introduce concept drift by gradually switching the function that defines the target variable between the 45,000th and the 55,000th sample. We also use the *LED* generator (Gordon et al., 1984), switching 5 of the 7 relevant features with irrelevant ones between instances 25000 and 75000. For the *Random Radial Basis Function* (RBF) datasets, we incorporate abrupt (RBF_a) or incremental (RBF_i) concept drift by switching or moving the centroids of the data distribution.

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

Table 2: Datasets used for experimental evaluations.

Type	Data Stream	Instances	Features	Classes
Synth.	Agrawal	100,000	9	2
	LED	100,000	24	10
	RBF_a	100,000	20	5
	RBF_i	100,000	20	5
Real	Covertype	100,000[4]	54	7
	Electricity	45,312	8	2
	Insects_a	52,848	33	6
	Insects_g	24,150	33	6
	Insects_i	57,018	33	6

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. In the following sections, we exclude the results for the RBF_i and Insects_i datasets, which can instead be found in Appendix B.

With respect to the neural network architecture, 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 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 based on what we found to perform well across several scenarios in our preliminary investigations. 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 refer to Appendix A for more detailed information on our experimental setup. We report our results as averages 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.

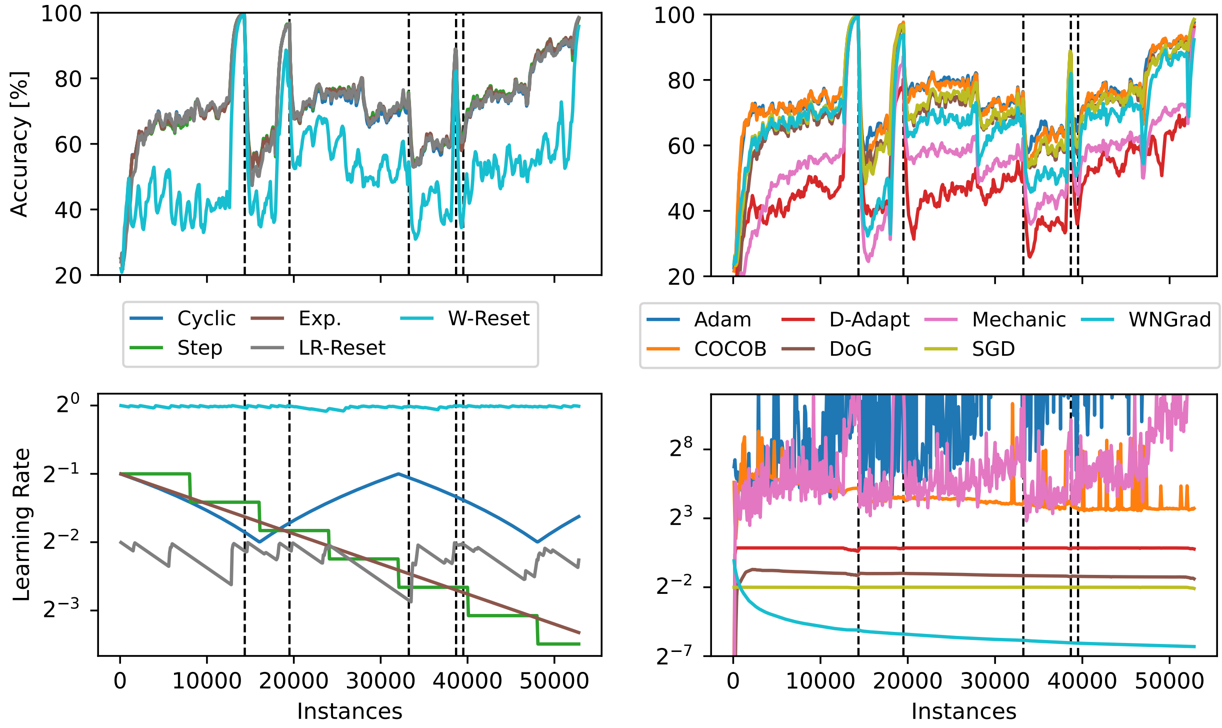
As seen in Table 3, our approach outperforms weight-resetting on all but Covertype but rarely yields an advantage over a basic exponential or fixed learning rate. Since both variants of our resetting approach perform almost identically, this performance gap is likely not caused by the drift detector. Instead, it appears that most concept drifts, even artificially created ones, are not severe enough for a drastic learning rate increase to be beneficial compared to the better stability of a more continuous learning rate progression. Overall, an appropriate fixed learning rate seems to be preferable to a decaying schedule with our drift resetting mechanism for the investigated data streams.

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 Covertype slightly surpasses the one achieved with a fixed learning rate but lags behind on most real-world 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 consistently outperforms its competitors. However, since SGD yields a high average accuracy on RBF_a but is clearly surpassed on Insects_a, the type of concept drift does not seem to be significant. Due to its favorable performance in terms of average accuracy and computational efficiency, SGD should be selected out of the non-parameter-free approaches in most cases. While some of the more elaborate optimizers like Adam or the SGD variant

[4] For Covertype we use only the first 100,000 from a total of 581,012 instances.



(a) Schedules

(b) Adaptive Optimizers

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

of Hypergradient Descent (HD) (Baydin et al., 2018) and WNGrad (Wu et al., 2020) also yield high accuracy scores in our experiments, they incur a more significant computational cost compared to SGD.

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 indicate 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.

Table 3: Average prequential accuracy [%] for static and drift adaptive learning rate schedules with SGD. LR-Reset$_A$ uses ADWIN, LR-Reset$_K$ uses Kolmogorov-Smirnov testing for drift detection. Best values are shown in **bold**, values within the 1σ interval of best values underlined.

Schedule	Agrawal	LED	RBF$_a$	Covertype	Electricity	Insects$_a$	Insects$_g$
Fixed	**79.78±1.09**	**73.91±0.04**	**89.57±2.61**	83.42±0.50	**73.77±0.40**	71.50±0.08	75.31±0.21
Exp.	77.98±1.36	73.38±0.14	87.66±3.53	82.95±0.26	73.51±0.48	72.19±0.37	**75.91±0.14**
Step	76.56±1.13	72.86±0.18	84.98±6.46	82.89±0.37	73.62±0.53	**72.23±0.27**	75.83±0.21
Cyclic	78.72±0.90	73.67±0.17	88.40±4.26	**83.44±0.08**	68.38±0.81	71.74±0.39	75.64±0.06
Weight-Reset	72.03±0.91	64.31±2.16	70.41±0.92	82.92±0.57	71.17±0.62	63.55±0.42	69.66±0.65
LR-Reset$_A$	78.26±0.73	72.95±0.13	87.60±3.43	82.92±0.32	73.07±0.66	71.48±0.34	75.54±0.11
LR-Reset$_K$	78.27±1.88	72.95±0.13	87.60±3.43	82.92±0.32	73.07±0.66	71.48±0.34	75.54±0.11

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

	Optimizer	Agrawal	LED	RBF_a	Covertype	Electricity	Insects_a	Insects_g
Tuned	SGD	79.64±2.03	73.91±0.04	89.57±2.61	**83.42±0.50**	73.77±0.40	71.50±0.08	75.31±0.21
	Adam	78.91±2.40	**73.98±0.20**	86.33±1.85	79.01±0.27	69.79±0.54	**75.38±0.24**	75.78±0.74
	AdaGrad	79.13±1.62	72.55±0.31	81.92±3.89	81.68±0.35	76.99±1.20	74.87±0.40	77.15±0.27
	WNGrad	76.91±0.47	64.66±0.34	80.07±0.67	76.98±0.15	70.80±0.59	66.25±0.19	66.75±0.40
	HD	**80.19±1.56**	73.81±0.09	89.78±3.37	83.36±0.25	73.83±0.32	70.67±0.06	73.37±0.21
LR-Free	COCOB	78.21±1.12	73.88±0.50	**90.75±1.28**	82.27±0.46	**84.48±0.88**	74.75±0.11	**77.67±0.17**
	DoG	78.77±2.25	73.34±0.13	87.04±3.13	83.07±0.64	71.53±0.70	70.59±0.26	74.01±0.21
	D-Adapt	60.16±0.96	54.69±8.34	41.37±3.34	76.69±0.79	66.03±1.75	50.05±11.26	48.21±10.62
	Mechanic	62.09±9.20	69.04±0.19	87.33±0.50	78.67±0.18	50.73±7.60	55.31±21.47	65.80±0.53

The best-performing Adam's learning rate 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 contribute significantly to Adam's high accuracy on Insects_a. Instead, it likely stems from its second-moment scaling, also featured in 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 effect of second-moment scaling 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.

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 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 and network architectures.

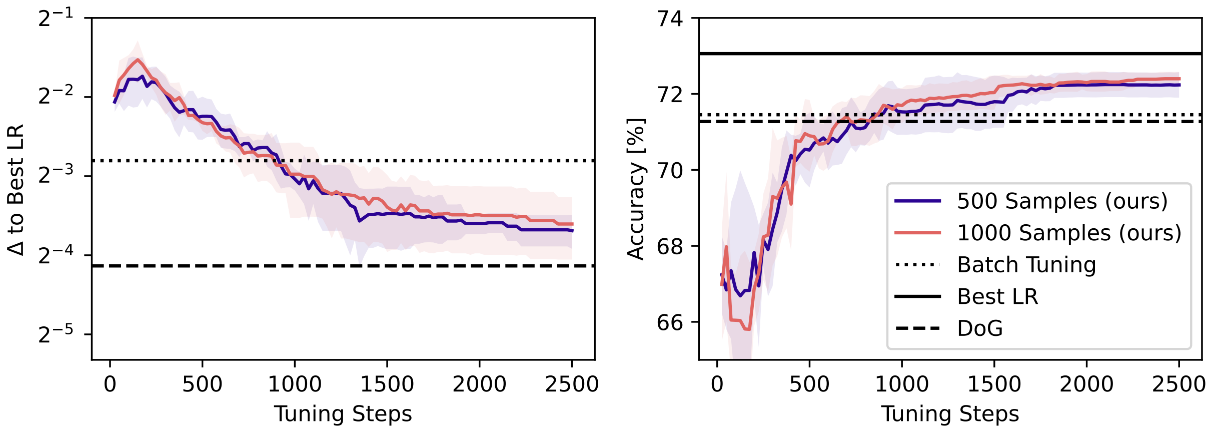


Figure 3: Absolute difference between pre-tuned and optimal learning rate and resulting accuracy on data streams for SGD and an exponential learning rate schedule with 500 or 1000 tuning samples. DoG is not included in first subplot since it is not intended for use with LR decay. Results are averaged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

Table 5: Prequential accuracy of learning rate tuning approaches averaged over all investigated MLP architectures. For Pre-Tuning$_{500}$ and Pre-Tuning$_{1000}$ we run our proposed learning rate tuning approach for 2000 steps with either 500 or 1000 samples and select the learning rate yielding the highest average prequential accuracy over the tuning run.

Approach	Agrawal	LED	RBF$_a$	Covertype	Electricity	Insects$_a$	Insects$_g$
SGD (Best LR)	83.70	73.95	93.84	83.14	74.15	71.98	75.28
COCOB	83.71±0.25	74.34±0.09	95.29±0.24	82.96±0.19	84.57±0.08	75.39±0.10	77.62±0.08
DoG	82.06±0.52	**73.71±0.11**	91.36±0.46	**82.56±0.15**	70.47±0.40	70.59±0.10	73.92±0.11
Batch Tuning	81.09±1.34	66.67±3.28	91.89±0.39	82.41±0.61	72.86±0.76	69.81±2.18	73.91±0.64
Pre-Tuning$_{500}$	82.01±0.32	73.26±0.03	**92.36±0.03**	80.25±1.86	73.34±0.45	**71.81±0.04**	**75.22±0.08**
Pre-Tuning$_{1000}$	**82.56±0.25**	73.30±0.00	92.18±0.38	80.80±1.05	**73.37±0.34**	**71.81±0.06**	75.17±0.05

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, demonstrating 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 approach's usage of the average prequential accuracy more accurately resembles the online learning process and 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 as a baseline due to being the best-performing parameter-free technique within the group of optimizers with a global learning rate that is the most comparable to our approach. This advantage comes despite a smaller average absolute deviation of DoG compared to our approach, which can be explained by the fact that DoG strongly underestimates the optimal learning rate at the beginning of training (see e.g. Figure 2b), where fast adaptation is crucial. Because DoG closely approaches the best learning rate for most of the stream, this barely affects its average deviation from the optimal learning rate.

The effectiveness of Pre-Tuning can also be seen in the dataset-specific results displayed in Table 5. Our tuning technique executed with 1000 samples (Pre-Tuning$_{1000}$) outperforms DoG and conventional tuning on all but the LED and Covertype datasets and often approaches the best learning rate determined via tuning directly on the testing stream (SGD Best LR). Even with 500 samples, the technique yields favorable results, indicating its usefulness in cases where data that is available pre-stream is scarce. While much more computationally expensive than the other evaluated approaches, the COCOB optimizer provides much better accuracy scores across all investigated data streams.

In conclusion, our proposed tuning approach enables significantly better learning rate selection for prequential evaluation on data streams than conventional tuning and DoG. Compared to learning-rate-free approaches like DoG or COCOB, Pre-Tuning also has the benefit that no additional memory or runtime costs are incurred once completed. This could be a critical advantage in streaming applications, where computing resources are often times a limiting factor. 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 limited in cases where the available tuning data is not representative of the subsequent data stream, such as 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 a theoretical background on discrepancies between learning rate optimization in conventional batch learning and online learning. Based on these differences, we propose a simple mechanism that resets a decayed learning rate on concept drift occurrences to increase the speed of adaption. Our empirical evaluation shows that our adaptation mechanism is much better suited when using neural networks than resetting the model weights, which is commonly done in conventional online learning. However, we find no advantage over using an appropriate fixed learning rate. We then experimentally compare adaptive optimization techniques that are popular for batch learning purposes in the context of online learning using multiple synthetic and real-world data streams. We find the DoG and especially the COCOB optimizer to yield high average accuracy scores even when compared to techniques requiring manual step size tuning. Lastly, we contribute a streaming-specific approach to learning rate tuning that grants significant performance increases over conventional tuning via a train-validation split and provides a valuable alternative to the computationally costly COCOB.

REFERENCES

R. Agrawal, T. Imielinski, and A. Swami. Database mining: A performance perspective. *IEEE Transactions on Knowledge and Data Engineering*, 5(6):914–925, December 1993. ISSN 1041-4347, 1558-2191, 2326-3865. doi: 10.1109/69.250074.

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.

Shibhansh Dohare, J. Fernando Hernandez-Garcia, Parash Rahman, Richard S. Sutton, and A. Rupam Mahmood. Loss of Plasticity in Deep Continual Learning, August 2023. URL http://arxiv.org/abs/2306.13812.

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.

Mohamed Elsayed and A. Rupam Mahmood. Utility-based Perturbed Gradient Descent: An Optimizer for Continual Learning, April 2023. URL http://arxiv.org/abs/2302.03281.

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.

A. D. Gordon, L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. Classification and Regression Trees. *Biometrics*, 40(3):874, September 1984. ISSN 0006341X. doi: 10.2307/2530946.

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.

Frank J. Massey. The Kolmogorov-Smirnov Test for Goodness of Fit. *Journal of the American Statistical Association*, 46(253):68–78, 1951. ISSN 0162-1459. doi: 10.2307/2280095.

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

Inyoung Paik, Sangjun Oh, Tae-Yeong Kwak, and Injung Kim. Overcoming Catastrophic Forgetting by Neuron-level Plasticity Control, July 2019. URL http://arxiv.org/abs/1907.13322.

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.

N.N. Schraudolph. Local gain adaptation in stochastic gradient descent. In *9th International Conference on Artificial Neural Networks: ICANN '99*, volume 1999, pp. 569–574, Edinburgh, UK, 1999. IEE. ISBN 978-0-85296-721-8. doi: 10.1049/cp:19991170.

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.

Richard S. Sutton. Adapting bias by gradient descent: An incremental version of delta-bar-delta. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, AAAI'92, pp. 171–176, San Jose, California, July 1992. AAAI Press. ISBN 978-0-262-51063-9.

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

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 Hyperparameter Settings for Empirical Evaluation

Table 6: Variable notation as used in this work.

Parameter	Symbol
Model Parameters	$\boldsymbol{\theta}$
Loss Function	L
Learning Rate	η
Learning Rate Decay Factor	γ
Drift Detection Confidence Level	δ
Steps Between LR Cycles/Updates	s
Relative LR at Midpoint of Cycle	$\hat{\eta}$

Table 7: Search spaces for learning rate of different optimizers.

Optimizer	Learning Rate Search Space
SGD	$2^1, 2^0, ..., 2^{-8}$
Adam	$2^{-3}, 2^{-4}, ..., 2^{-12}$
AdaGrad	$2^1, 2^0, ..., 2^{-8}$
WNGrad	$10^{1.25}, 10^{0.75}, ..., 10^{-7.75}$
HD	$2^{-3}, 2^{-4}, ..., 2^{-12}$
COCOB	100
DoG	1
D-Adapt	1
Mechanic	0.01

Table 8: Configuration of other parameters.

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

B Full Experimental Results

In this section, we provide our full experimental results including the RBF_i and Insects_i datastreams as well as a detailed

B.1 Architecture-specific Learning Rate Tuning Results

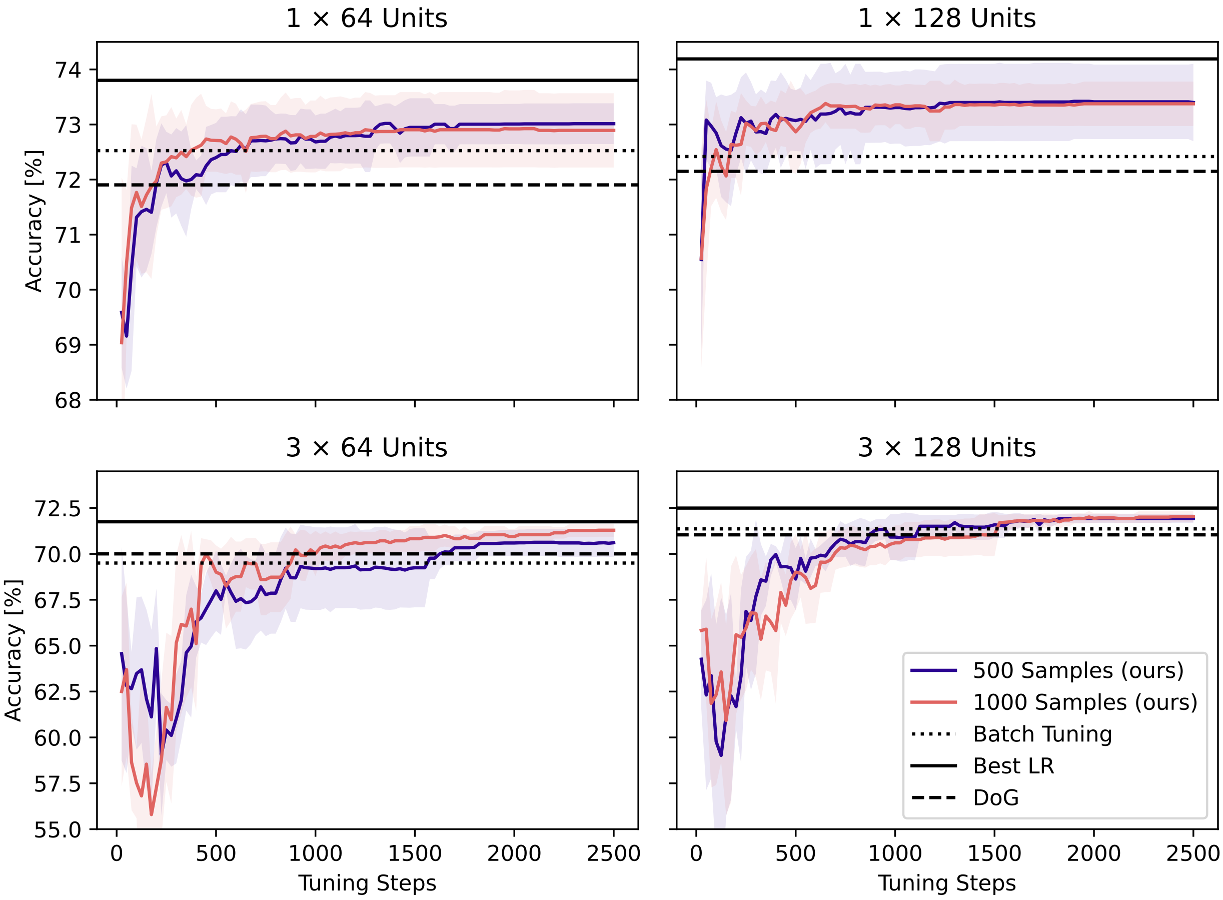


Figure 4: Average accuracy over all evalutated real-world datasets achieved by Pre-Tuning for different network architectures.

B.2 Full Results on Learning Rate Tuning

Table 9: Full prequential accuracy results of learning rate tuning approaches averaged over all investigated MLP architectures. For Pre-Tuning$_{500}$ and Pre-Tuning$_{1000}$ we run our proposed learning rate tuning approach for 2000 steps with either 500 or 1000 samples and select the learning rate yielding the highest average prequential accuracy over the tuning run.

Approach	Agrawal	LED	RBF$_a$	Covertype	Electricity	Insects$_a$	Insects$_g$	Insects$_i$	RBF$_i$
Oracle	83.70	73.95	93.84	83.14	74.15	71.98	75.28	60.75±10.00	81.52±10.00
COCOB	83.71±0.25	74.34±0.09	95.29±0.24	82.96±0.19	84.57±0.08	75.39±0.10	77.62±0.08	**64.02±.11**	78.59±.38
DoG	82.06±0.52	**73.71±0.11**	91.36±0.46	**82.56±0.15**	70.47±0.40	70.59±0.10	73.92±0.11	58.83±.07	**77.63±1.24**
Batch Tuning	81.09±1.34	66.67±3.28	91.89±0.39	82.41±0.61	72.86±0.76	69.81±2.18	73.91±0.64	58.27±2.21	72.51±1.63
Pre-Tuning$_{500}$	82.01±0.32	73.26±0.03	**92.36±0.03**	80.25±1.86	73.34±0.45	**71.81±0.04**	**75.22±0.08**	60.55±.10	73.19±.31
Pre-Tuning$_{1000}$	**82.56±0.25**	73.30±0.00	92.18±0.38	80.80±1.05	**73.37±0.34**	**71.81±0.06**	75.17±0.05	60.33±.28	72.98±.44

B.3 Full Learning Rate Scheduling Results

Table 10: Full results on average prequential accuracy [%] for static and drift adaptive learning rate schedules with SGD. LR-Reset$_A$ uses ADWIN, LR-Reset$_K$ uses Kolmogorov-Smirnov testing for drift detection. Best values are shown in **bold**, values within the 1σ interval of best values underlined.

Schedule	Agrawal	LED	RBF$_a$	Covertype	Electricity	Insects$_a$	Insects$_g$	Insects$_i$	RBF$_i$
Fixed	**79.78±1.09**	**73.91±0.04**	**89.57±2.61**	83.42±0.50	**73.77±0.40**	71.50±0.08	75.31±0.21	60.48±.20	57.18±2.69
Exp.	77.98±1.36	73.38±0.14	87.66±3.53	82.95±0.26	73.51±0.48	72.19±0.37	**75.91±0.14**	**61.28±.16**	50.75±1.38
Step	76.56±1.13	72.86±0.18	84.98±6.46	82.89±0.37	73.62±0.53	**72.23±0.27**	75.83±0.21	**61.18±.11**	50.69±1.93
Cyclic	78.72±0.90	73.67±0.17	88.40±4.26	**83.44±0.08**	68.38±0.81	71.74±0.39	75.64±0.06	60.48±.20	58.67±1.63
Weight-Reset	72.03±0.91	64.31±2.16	70.41±0.92	82.92±0.57	71.17±0.62	63.55±0.42	69.66±0.65	49.97±.67	**61.07±.76**
LR-Reset$_A$	78.26±0.73	72.95±0.13	87.60±3.43	82.92±0.32	73.07±0.66	71.48±0.34	75.54±0.11	60.39±.18	52.88±1.72
LR-Reset$_K$	78.27±1.88	72.95±0.13	87.60±3.43	82.92±0.32	73.07±0.66	71.48±0.34	75.54±0.11	60.39±.18	52.88±1.72

B.4 Full Results on Adaptive Optimization Techniques

Table 11: Full results on average prequential accuracy [%] for different optimizers. Best values are shown in **bold**, values within the 1σ interval of best values underlined.

	Optimizer	Agrawal	LED	RBF_a	Covertype	Electricity	Insects_a	Insects_g	Insects_i	RBF_i
Tuned	SGD	79.64±2.03	73.91±0.04	89.57±2.61	**83.42±0.50**	73.77±0.40	71.50±0.08	75.31±0.21	60.48±.20	**57.18±2.69**
	Adam	78.91±2.40	**73.98±0.20**	86.33±1.85	79.01±0.27	69.79±0.54	**75.38±0.24**	75.78±0.74	**64.17±.13**	**60.32±3.75**
	AdaGrad	79.13±1.62	72.55±0.31	81.92±3.89	81.68±0.35	76.99±1.20	74.87±0.40	77.15±0.27	62.51±.59	45.00±1.55
	WNGrad	76.91±0.47	64.66±0.34	80.07±0.67	76.98±0.15	70.80±0.59	66.25±0.19	66.75±0.40	56.14±.21	42.06±.43
	HD	**80.19±1.56**	73.81±0.09	89.78±3.37	83.36±0.25	73.83±0.32	70.67±0.06	73.37±0.21	59.92±.18	56.37±5.01
LR-Free	COCOB	78.21±1.12	73.88±0.50	**90.75±1.28**	82.27±0.46	**84.48±0.88**	74.75±0.11	**77.67±0.17**	63.93±.17	51.70±2.11
	DoG	78.77±2.25	73.34±0.13	87.04±3.13	83.07±0.64	71.53±0.70	70.59±0.26	74.01±0.21	59.66±.22	55.72±2.43
	D-Adapt	60.16±0.96	54.69±8.34	41.37±3.34	76.69±0.79	66.03±1.75	50.05±11.26	48.21±10.62	36.00±11.81	42.61±1.46
	Mechanic	62.09±9.20	69.04±0.19	87.33±0.50	78.67±0.18	50.73±7.60	55.31±21.47	65.80±0.53	47.89±17.46	44.45±.85