# Learning Rate Optimization for Online Deep Learning

**Anonymous submission**


## Abstract

The efficient training via gradient-based optimization techniques is an essential building block of the success of deep learning. These techniques have been enabled in part by extensive research on the impact and the effective estimation of an appropriate learning rate. Despite the proliferation of data streams produced by IoT devices, digital platforms etc., previous research has been largely focused on batch learning, which assumes that all training data is available at once. However, because the gradual emergence and non-stationarity of data, as well as other characteristics, present additional challenges, the insights from 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 common 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.


## Introduction

Deep learning models have demonstrated exceptional performance in various domains, with the choice of optimizer playing a crucial role in achieving outstanding results. In the context of batch learning, where all data is available simultaneously, extensive research has been conducted to explore different optimizer choices and optimization techniques for deep learning architectures. Numerous methods have emerged to effectively update the weights of these architectures. However, the investigation of optimizer choices in online learning, where models must adapt to evolving data streams, remains relatively limited.

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. Specifically, we address the following research questions:

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

LR tun-
graphic
?

For the first research question, we explore how the selection of an optimizer differs when moving from the traditional batch learning setting to the dynamic online learning scenario. We examine the suitability of various optimizer choices in online learning and their impact on model performance.

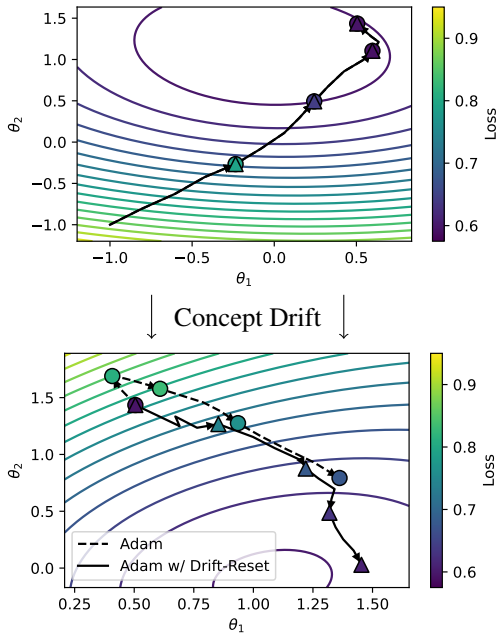


Figure 1: Parameter trajectory of Adam (Kingma and Ba 2017) with or without adaptation to concept drift on synthetic data stream with abrupt concept drift. Marker colors depict the expected prequential loss over the last 16 data instances.

The second research questions investigates practical choices for gradient-based online training of deep architectures. We analyze different optimization techniques and explore their effectiveness in adapting to evolving data streams while maintaining model performance. The third research question focuses on the performance of adaptive methods in online deep learning scenarios. These methods dynamically adjust the learning rate based on gradient characteristics, allowing models to adapt more effectively to chang-

ing data patterns. We compare the performance of adaptive methods against other optimization approaches to determine their suitability for online deep learning tasks. Through our in-depth analysis and experimentation, we aim to enhance our understanding of optimizer choices in online deep learning. By shedding light on the impact of optimizers, learning rates, and batch sizes, and comparing the effectiveness of adaptive methods, we aim to enable researchers and practitioners to make informed decisions when selecting optimization techniques for real-time learning tasks.

## Learning Rate Scheduling

In the following, we will briefly outline the most important differences between the influence of the learning rate of first order gradient-based optimization methods in streaming- and in conventional batch learning environments.

First order gradient-based optimization approaches like stochastic gradient descent and its derivatives aim to iteratively minimize the error of a DL model using only first order gradient information at each step $t$. We denote the gradient of the prediction error for a mini-batch of training samples $y_t, X_t \sim p_t$ with respect to model-parameters $\theta$ as

$$g_t(\theta) = \nabla_\theta \mathcal{L}(y_t, f(X_t; \theta)), \tag{1}$$

where $\mathcal{L}$ represents a loss function (e.g., cross-entropy for classification- or mean squared error for regression tasks). Using this notation, the update performed by SGD with a constant learning rate $\eta$ at each iteration $t$ is given by

$$\theta_{t+1} = \theta_t - \eta \cdot g_t(\theta_t). \tag{2}$$

Many previous works deal with the intricacies of SGD in the context of training deep architectures in a batch learning setting (see Bengio (2012); Bottou (2012); Goodfellow, Bengio, and Courville (2016)). The primary trade-off identified by these works when it comes to the selection of an appropriate learning rate is that between the speed of convergence and the amount of stochasticity. While increasing the learning rate speeds up convergence, it also increases stochasticity and therefore leads to the divergence of the training criterion beyond a certain threshold. Smith and Le (2018) for instance, found that when modelling SGD as a stochastic differential equation, the "noise scale" is directly tied to $\eta$ (Smith and Le 2018). In biological terms, increasing the learning rate increases plasticity, whereas decreasing it increases stability.

To get the best out of large and small values for $\eta$, a learning rate schedule $(\eta_1, \ldots, \eta_T)$, which assigns a learning rate to each update step $t \in 1, \ldots, T$ can be used. It is for instance common to set a high learning rate initially and decrease it throughout the training process. This ensures fast convergence and a higher level of noise at the start of training, which has been claimed to help SGD skip over sharp minima with poor generalization (Hochreiter and Schmidhuber 1997; Chaudhari et al. 2017), while mitigating jumping around potential minima at later stages. 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 simple way to achieve such a schedule is to exponentially decay $\eta$ after each update by multiplying it with a factor $\gamma < 1$, although some practitioners have come to prefer schedules with sharper drop-offs like step schedules that decrease $\eta$ by a larger margin after a certain number of updates (Smith et al. 2018). Other popular options include cyclic learning rate schedules which oscillate $\eta$ between a base- and a maximum or minimum value over a predefined interval. 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 (Kingma and Ba 2017). Wu et al. (2019) provide a detailed analysis on the effect of learning rate policies like the aforementioned ones.

In contrast to the field of conventional batch learning, the impact of the learning rate in stream-based deep learning is a lesser studied issue. 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.

These requirements lead to significant differences with respect to the problem of learning rate optimization compared to batch learning. Nevertheless, only few studies on the impact and tuning of the learning rate in stream-based deep learning exist.

In batch learning, the task of finding an optimal schedule for $\eta$ can be defined as

$$\begin{aligned} &\min_{\eta_0,\ldots,\eta_T} \sum_{i=1}^{V} \mathcal{L}(y_i, f(X_i; \theta_T)) \\ &\text{s.t.} \qquad X_i, y_i \sim p^{(v)} \qquad\qquad \forall i \in 1, \ldots, V, \end{aligned} \tag{3}$$

where $p^{(v)}$ is a distribution of validation data, usually made up of a dataset split off from the training dataset. Verbally, learning rate optimization in batch learning aims to find a schedule $(\eta_0, \ldots, \eta_T)$ leading to parameter values $\theta_T$ at the end of a training run that in turn minimize the prediction error for validation data.

Under the requirements described above, however, we must define the task differently as

$$\begin{aligned} &\min_{\eta_0,\ldots,\eta_T} \sum_{t=0}^{T} \mathcal{L}(y_t, f(X_t; \theta_{t-1})) \\ &\text{s.t.} \qquad X_t, y_t \sim p_t \qquad\qquad \forall t \in 1, \ldots, T. \end{aligned} \tag{4}$$

Compared to Problem (3), the most apparent difference is that there is no separate validation data. Instead, due to Requirement , the goal is to minimize $\mathcal{L}$ with respect to the next instances $X_{t+1}, y_{t+1}$ at each timestep $t$.

This means that in contrast to Problem (3), not only the final parameters $\theta_T$ but every parameter configuration $\theta_t$ in the entire trajectory contributes equally to the objective. Therefore, speed of convergence is of much larger importance in the streaming setting, whereas the performance of

the final parameters $\theta_T$ has relatively little impact. Since memory is limited (Requirement ), 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 sharp minima can be suboptimal with respect to Problem 4 even if it eventually yields a lower loss. Another difference to conventional batch learning is that the distribution $p_t$ of the data stream might, and in practice most likely will, be subjected to change in the form of concept drift over time. Under such circumstances, the optimal parameter values $\theta^*$ move throughout the progression of the stream increasing the distance to the model parameters.

about reti- conver- e bound kosky, zio, and ta 2023)

Since the theoretically optimal learning rate $\eta^*$ is proportional to the quadratic distance between initial and optimal parameters $||\theta_1 - \theta^*||^2$ (albeit under some constraints like the absence of noise) (Carmon and Hinder 2023), it should be larger when concept drift occurs.

We therefore propose a simple adaptation to decaying learning rate schedules that operates in a fixed value range and restore the model's ability to adapt to changes when needed by resetting $\eta$ to its original value if a concept drift has occurred. To this end we apply an ADWIN drift detector (Bifet and Gavaldà 2007) to the underlying model's prequential loss to reduce the drift detection task to a computationally less expensive univariate problem. While our approach does not allow for a continuous adaptation like the technique by Kuncheva and Plumpton (2008) and might therefore be less suited for more subtle drift, it offers more reliability since the initial learning rate cannot be exceeded.

about - ability ndent re- ng with SWIN?

Concept drift also complicates the tuning of $\eta$, 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, that aims to approximately solve Problem 4.

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 $\eta$ 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 .

## Adaptive Optimizers

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.

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.

For instance, 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 $\eta$ (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). Even though Schaul, Zhang, and LeCun (2013) demonstrate *vSGD's* robustness to non-stationary data distributions, it has, to the best of our knowledge, not been widely adopted in the online learning space.

Did not e uate vSG since its and there no imple tation. Is valid?

Instead of using curvature information for adapting $\eta$, 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).

-SGDHD: Meta learning approach optimizing the learning rate itself also using SGD -WNGrad: estimates lipschitz constant of gradient of loss function to derive learning rate $\eta$ -Mechanic: can wrap around any first order algorithm, removing the need of tuning $\eta$, uses a base online convex optimization (OCO) algorithm as well as a meta OCO to optimize the learning rate with respect to the theoretical upper convergence bound of SGD -DoG: also optimizes theoretical upper bound by estimating $||\theta_0 - \theta^*||$ as $\max_{i<t} ||\theta_0 - \theta_i||$ -DAdapt: modification of the AdaGrad step size applied to weighted dual averaging (Duchi, Agarwal, and Wainwright 2012), together with our key innovation: D lower bounding

Furthermore, several studies developed paramater-free optimizers for specific areas of application such as time series forecasting (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). Due to our focus for the present work being general data stream applications, we did not further investigate these techniques either.

Despite the fact that this family of optimizers is inherently well-suited for the highly non-stationary streaming data (Schaul, Zhang, and LeCun 2013), their application on this kind of data has rarely been investigated. Besides Schaul, Zhang, and LeCun (2013) none of these optimizers target online learning scenarios and therefore do not explicitly consider concept drift, which raises the question of whether they are suitable for stream-based learning. Previous work on parameter-free optimization of online deep learning models is rather limited, and mostly focused on specific applications.

| Optimizer | Runtime | Space | Param. specific | LR free |
|---|---|---|---|---|
| DAdapt | $\mathcal{O}(6D)$ | $\mathcal{O}(2D)$ | ✗ | ✓ |
| DoG | $\mathcal{O}(5D)$ | $\mathcal{O}(1D)$ | ✗ | ✓ |
| Mechanic | $\mathcal{O}(10D)$ | $\mathcal{O}(1D)$ | ✗ | ✓ |
| WNGrad | $\mathcal{O}(2D)$ | $\mathcal{O}(0)$ | ✗ | ✓ |
| SGDHD | $\mathcal{O}(2D)$ | $\mathcal{O}(1D)$ | ✗ | ✓ |
| COCOB | $\mathcal{O}(14D)$ | $\mathcal{O}(4D)$ | ✓ | ✓ |
| Adam | $\mathcal{O}(12D)$ | $\mathcal{O}(2D)$ | ✓ | ✗ |
| vSGD | $\mathcal{O}(21D)$[1] | $\mathcal{O}(4D)$ | ✓ | ✓ |
| AdaGrad | $\mathcal{O}(5D)$ | $\mathcal{O}(1D)$ | ✓ | ✗ |

Table 1: Overview of additional time- and space-complexity of adaptive first-order optimizers compared to basic SGD. Values are given in big O notation with respect to the number of model parameters $D$. We do not list convergence guarantees because the guarantees given in the original papers of different optimizers are based on different assumptions and are rarely compatible with streaming applications.

## Experiments

| Type | Data Stream | Samples | 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 incr.-grad. | 24150 | 33 | 6 |
| | Covertype[2] | 100000 | 54 | 7 |
| | Electricity | 45312 | 8 | 2 |

Table 2: Datasets used for experimental evaluations.

[1]Complexity for feed-forward neural networks. Since *vSGD* requires additional backpropagation steps, its complexity is architecture dependent.

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

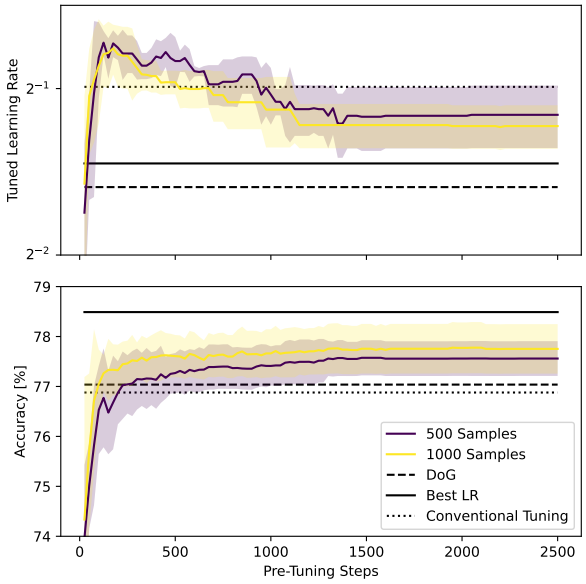


Figure 2: Pre-tuned LR (LR that maximizes accuracy on pre-tuning data) and resulting accuracy on data streams when using SGD and an exponential learning rate schedule with 500 or 1000 separate tuning samples. Results are averaged over all real-world datasets. The shaded area represents the $1\sigma$-interval.

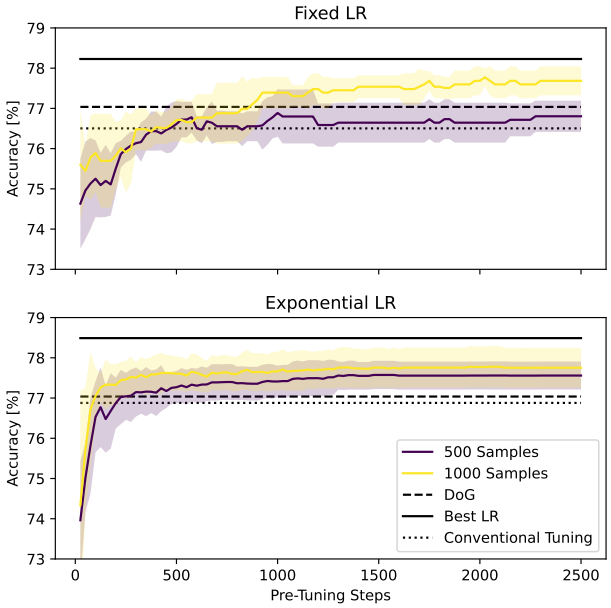


Figure 3: Accuracy achieved by pre-tuning on 500 or 1000 samples when using SGD with a fixed LR schedule (top) or an exponential schedule (bottom), averaged over all real-world datasets. The shaded area represents the $1\sigma$-interval.

| Optimizer | Schedule | RBF abrupt | RBF incr. | Covertype | Insects abrupt | Electricity |
|---|---|---|---|---|---|---|
| SGD | Fixed | 93.70±.90 | 69.33±1.36 | 83.08±.18 | 71.12±.08 | 73.12±.42 |
| | Exp. | 94.00±.52 | 69.24±1.06 | 82.65±.19 | 71.32±.19 | 73.06±.42 |
| | Exp. Reset | 94.28±.37 | 69.76±.92 | 82.70±.27 | 71.27±.14 | 73.05±.45 |
| | Step | 94.00±.68 | 69.16±.99 | 82.74±.10 | 71.39±.17 | 72.96±.48 |
| | Step Reset | 94.04±.70 | 69.75±.81 | 83.03±.13 | 71.19±.14 | 73.18±.50 |
| | Cyclic | 94.45±.25 | 73.72±1.16 | **83.40±.21** | 71.41±.20 | 67.80±1.03 |
| | Cyclic Reset | **94.50±.21** | **73.78±1.15** | 83.33±.13 | 71.39±.15 | 67.83±1.00 |
| Adam | Fixed | 92.77±.41 | 66.46±4.39 | 78.85±.22 | **75.08±.13** | 69.23±.41 |
| | Exp. | 92.17±.82 | 63.91±3.52 | 78.53±.27 | 74.88±.15 | 69.33±.40 |
| | Exp. Reset | 92.83±1.19 | 62.03±2.59 | 77.05±.08 | 72.04±.49 | 68.31±.40 |
| AdaGrad | Fixed | 91.34±.83 | 50.39±3.60 | 81.07±.22 | 74.31±.34 | 76.64±1.92 |
| SGDHD | Fixed | 91.03±.45 | 63.38±1.55 | 82.33±.12 | 67.35±.16 | 73.10±.10 |
| COCOB | Fixed | 93.40±.38 | 63.52±2.70 | 82.27±.46 | 74.75±.11 | **84.30±.56** |
| WNGrad | Fixed | 87.23±1.24 | 44.79±.76 | 76.95±.15 | 66.14±.15 | 70.74±.59 |
| DAdaptSGD | Fixed | 74.91±4.22 | 45.47±2.75 | 76.69±.79 | 50.05±11.26 | 66.03±1.75 |
| DoG | Fixed | 92.73±.59 | 73.17±2.72 | 83.07±.64 | 70.59±.26 | 71.53±.70 |

Table 3: Average prequential accuracy [%] for the three best learning rates.

| Resetting Approach | RBF abrupt | RBF incr. | Covertype | Insects abrupt | Electricity |
|---|---|---|---|---|---|
| ADWIN (Two Tailed) | **94.28±.37** | 69.76±.92 | 82.70±.27 | 71.27±.14 | 73.05±.45 |
| ADWIN (One Tailed) | 94.25±.38 | 70.22±2.58 | 82.64±.20 | 71.25±.15 | 73.00±.50 |
| ADWIN Weight Reset | 71.79±.73 | 65.07±.31 | 82.54±.16 | 50.97±.36 | 70.19±1.00 |
| KSWIN | 94.23±.55 | 70.10±1.98 | **83.01±.06** | **71.38±.16** | 73.13±.31 |
| P-KSWIN | 93.86±.48 | **70.71±1.38** | **83.01±.18** | 71.25±.15 | **73.26±.43** |

Table 4: Average prequential accuracy [%] for the three best learning rates.

We ran prequential evaluations using basic SGD with variable batch sizes and learning rates for synthetic data streams with and without incremental concept drift, the results of which are displayed in Figure 5. For static data, the average prequential accuracy over the entire stream gradually improves when moving up from an inadequately low learning rate until a certain point where training begins to diverge and performance consequently crashes. Based on our results, there seems to be an inverse relationship between batch size and both the optimal learning rate and the optimal accuracy, with larger batch sizes seemingly increasing the risk of divergence.

(5)

This effect is much stronger in the presence of concept drift as the results for RBF Incremental show.

¡-could be explained by the fact that the presence of concept drift exacerbates the gradient stochasticity caused by the delay between observation and learning of samples.

## Conclusion

## References


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

Bottou, L. 2012. Stochastic Gradient Descent Tricks. In Montavon, G.; Orr, G. B.; and Müller, K.-R., eds., *Neural Networks: Tricks of the Trade: Second Edition*, Lecture Notes in Computer Science, 421–436. Berlin, Heidelberg: Springer. ISBN 978-3-642-35289-8.

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.

Carmon, Y.; and Hinder, O. 2023. Making SGD Parameter-Free. arxiv:2205.02160.

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 Anal-

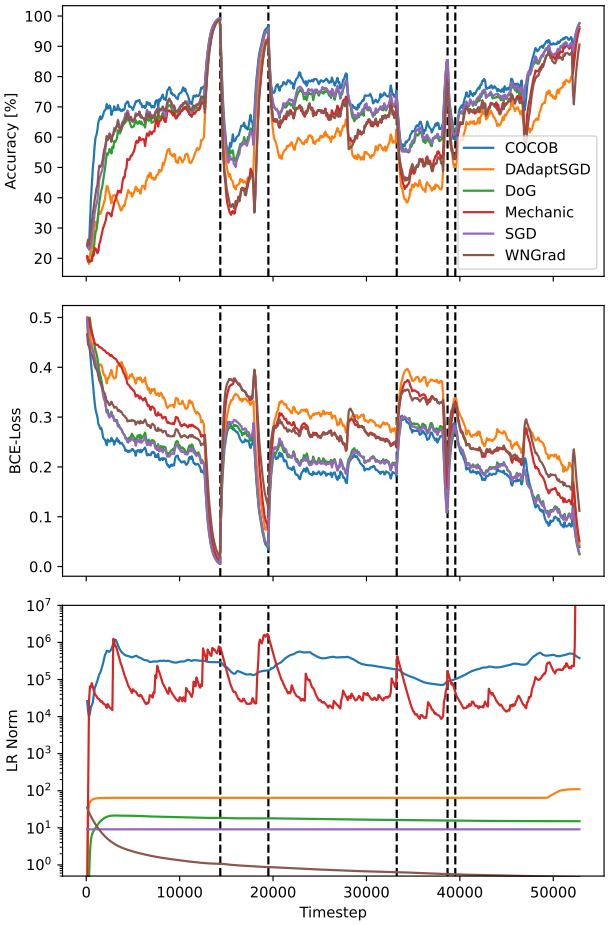


Figure 4: Prequential accuracy, binary cross-entropy loss and LR norms $||\eta_t||$ over time for various optimization algorithms on Insects abrupt. Each dashed vertical line represents a concept drift. Lines are exponentially smoothed with a factor of 0.8.

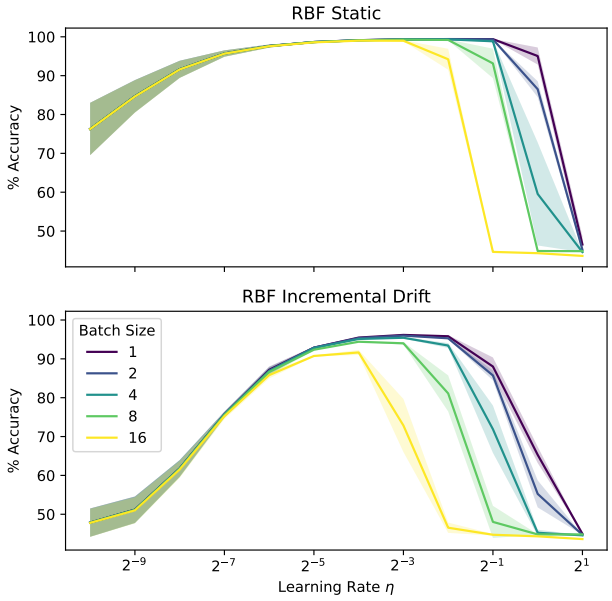


Figure 5: Average accuracy over static and incrementally drifting synthetic data streams in relation to SGD mini-batch size and learning rate $\eta$. Shaded areas mark the $1\sigma$ interval.

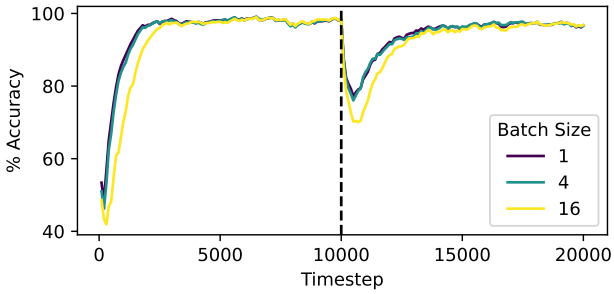


Figure 6: Prequential accuracy for different batch sizes on RBF abrupt data stream.

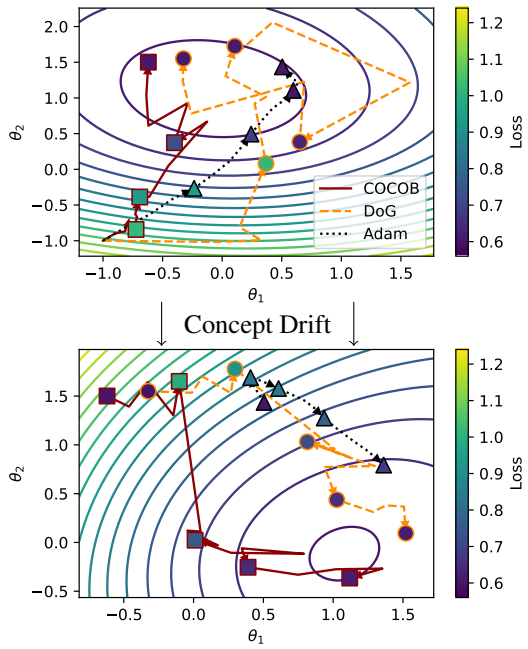


Figure 7: Parameter trajectory of COCOB (Orabona and Tommasi 2017), DoG (Ivgi, Hinder, and Carmon 2023) and Adam (Kingma and Ba 2017) on synthetic data stream with abrupt concept drift. Marker colors depict the expected prequential loss over the last 16 data instances.


ysis 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.

Goodfellow, I.; Bengio, Y.; and Courville, A. 2016. *Deep Learning*. MIT Press.

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.

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

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

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.

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.

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.