Responsible AI: Training deep learning model efficiently

Shwetha Salimath[0000−0001−9213−3780], Francesca Bugiotti[0000−0002−6555−9652], and Sylvain Wlodarczyk

[1] Universite-Paris-Saclay, Gif-sur-Yvette, France
[2] SLB, Montpellier, France

Abstract. There has been a significant surge in the adoption of deep learning methods across various industrial and research domains. This increase requires efficient training strategies to improve convergence speed, robustness, and overall model optimization. The paper presents a benchmark analysis that highlights the benefits of learning rate (LR) schedulers. The benchmark has been conducted using four training methods that integrate the LR finder, the LR scheduler, and the LR optimizers (Adam). The first three training methods use the LR finder to compute the correct range and then use, respectively, the AdaptiveCycle LR scheduler, the OneCycle LR scheduler, and the static LR for training; the last training method uses only a static LR. The OneCycle LR scheduler has been used as it balances exploration and exploitation during training by adjusting the LR cyclically. Additionally, we introduce a novel scheduling method, AdaptiveCycle, that modifies the LR based on validation loss plateaus over a cyclic LR curve, further optimizing the training process. We have performed the benchmark using diverse time-series data sets from the UCR and UCE repositories. The benchmark is performed across eleven deep learning models, simple such as LSTM, FCN, and complex models such as Inception-time, XCM, and LSTM-FCN. The accuracy, F1 score, and computational time allow us to highlight the adaptability of training methods across various datasets and models. The findings emphasize the advantages of employing LR schedulers for efficient training. The AdaptiveCycle scheduler achieves optimal performance across different models and datasets. The benchmark results will encourage the adoption of the LR scheduler in research and industry.

Keywords: Deep Learning · Responsible AI · Training framework · Benchmark · LR schedulers

1 Introduction

The use of deep learning has increased significantly in the past decade across various applications. It has revolutionized the fields of computer vision, NLP, and autonomous systems. The two main reasons for this are the advancement in GPUs [16] and the enormous amount of data available [21]. Current technology

allows us to collect, store, and process vast amounts of data to provide insights and uncover complex data patterns using deep learning. With GPUs being accessible and affordable, they have played a pivotal role in enabling researchers and developers to experiment with deep learning technologies.

With the increasing usage of neural networks, it is important to apply optimal training dynamics for their efficient training. Training dynamics refers to the process by which the neural networks update the weights associated with neurons in different layers of the model during training. Optimizing this process would lead to faster model convergence, saving computational time and resources while not compromising model robustness [1]. The learning rate (LR) is the most important hyperparameter in the neural network training process [11]. The LR is used to control the step size for weight updates during model training. A too high LR may lead to failed convergence due to unstable training, while a too low LR requires many iterations to converge, leading to slow training [15].

Three main strategies are available to set the LR: LR finder, which helps to select the best LR range for training; adaptive learning rate methods for optimization, such as Adam [13], RMSprop, and LR schedulers to dynamically change it during training. In this paper, we have benchmarked the results of five different scenarios of combinations of the above methods. The main focus of this paper is to understand that a static LR for training is disadvantageous, as it fails to adapt the training process according to the optimization process. In optimal training, two subprocesses occur: exploration and exploitation. We need to explore the best model weights for convergence; for this, a high LR is required at the start of the training. Then, we need to fine-tune the model weight for an optimal solution, which is referred to as exploitation, and it requires a low LR. Thus, a static LR cannot satisfy both conditions, and a compromise needs to be made.

As stated before, we can use an adaptive optimizer like Adam, which will dynamically adjust the weight for each parameter according to the gradient information. This is the most widely practiced solution. The second is to use LR schedulers; although they have gained popularity, they are still not widely used. The popular LR schedulers are Exponential decay, Step decay, Cosine annealing, and Cyclic Learning rates [10]. This paper focuses more on the One Cycle policy [18] which does not follow a gradually decreasing LR unlike the other schedulers. In OneCycle, the LR first starts small, gradually increases for exploration during high LR and then finally starts decreasing back to the lower LR for fine-tuning the model during exploitation. The LR change process is almost identical to that required for optimal training. OneCycle also adjusts the momentum for the adaptive optimizers to counter the effect of the varying LR, which might make the model unstable. More details on these schedulers are provided in Section 2, explaining the existing related work.

In this we introduce a new scheduler called AdaptiveCycle that could adopt the LR depending on validation loss during training. AdaptiveCycle uses the OneCycle policy on the backhand, which allows us to adjust the LR if the validation loss has been increasing or on a plateau. Section 4, on methodology,

explains in detail the algorithm of AdaptiveCycle to dynamically update the LR. The dependence on validation loss allows the model to overfit. While doing a literature review, we found the limited benchmark studies conducted in this domain across diverse tasks, especially across functions with varying data characteristics, and imbalanced and noisy datasets. The datasets used are time series datasets from the UCR/UCE, having different features, as stated in Section 4, benchmark setup. Time-series data were chosen to perform the benchmark experiments as it is used in many industries, such as Finance, Economics, Healthcare, Energy, Telecommunications, Retail, Logistics, Manufacturing, and Environment. The adaptability is tested across different deep learning models with various sizes and complexities. The models used are We have evaluated the performance across four main categories: accuracy, convergence speed, stability, and model size. Brief explanations of the models used are provided in Section 4, the benchmark setup.

Section 5 consists of the results and analysis. The conclusion and recommendations of best practices are stated in Section 6. The main goal of the paper is to provide the advantages of LR schedulers and to encourage their usage among researchers and industry professionals, allowing more efficient training and a robust, well-generalized model.

2 Related Work

This section gives a brief overview of the different adaptive optimizers and LR schedulers that are available. Stochastic gradient descent (SGD) is a foundation optimizer that uses gradients (first moments) to update parameters using a randomly selected data subset. Unlike traditional optimizers such as SGD, adaptive optimizers modify the LR for each parameter using squared gradients (second moment). Adgrad [5] updates the LR based on the historic sum, while RMSprop uses the moving average of the squared gradients. Adam (adaptive moment estimator) uses an exponential moving average of both the moments. Adam is the most widely used as it is computationally efficient, requires less memory, and works well with large models and datasets.

The StepLR, Cosine annealing, and OneCycle are the most widely used LR schedulers [10]. Step LR reduces the LR by a fixed factor after a specified number of epochs, called the step size. It is simple with a gradual LR decay and is effective for model training, especially for fine-tuning. Cosine annealing reduces the LR following a cosine curve. It initially starts from a high LR and gradually approaches the minimum LR, and a warm restart is performed at the end of training. It usually provides a smooth loss decay, but still needs a larger number of epochs. The warm restart helps escape local minima.

OneCycle adjusts the LR cyclically over one smooth cycle [19]. Initially, the LR starts at an assigned minimum value to warm up. Then it gradually increases to the maximum value; having a high LR accelerates training. Finally, the LR decreases back to the initial minimum LR or even to a slightly lower LR for fine-tuning. The momentum is also adjusted inversely to the LR rate

curve to balance the impact of high and low LRs. The high LR regions result in the exploration of global minima and help avoid local minima. This allows the model to avoid overfitting with better generalization and faster convergence. OneCycle is used prominently for computer vision and NLP tasks, especially for large datasets and models.

Studies have highlighted that schedulers like Cosine Anneling and OneCycle improve the convergence rate, especially with OneCycle, which creates balanced training for large datasets and models. However, most of these studies were conducted on image datasets such as MNIST and CIFAR, over just a few models. There is a lack of understanding of how it performs on smaller datasets and models [22].

3 Benchmark setup

This section elaborates on the dataset employed, emphasizing its various features. Additionally, an outline of the models applied in the research is presented, along with five training scenarios. Details on the evaluation metrics and the experimental setup are also provided.

3.1 Datasets

We primarily used the existing UCR benchmark dataset, as our results can be compared with other benchmarks. We have selected four univariate and four multivariate datasets from both UCR and UCE [3]. Table 1 illustrates these datasets, provides a detailed summary of various datasets, and highlights key attributes for each. In particular, we organized these signals into distinct categories based on the origin and nature of the time series.

Most of the parameters of deep learning models depend on time series attributes, such as input length, input features, and the number of output classes and labels. Hence, we also group the datasets according to characteristics that may impact the formation and performance of the model. Each dataset is characterized by several attributes: Length, Strength, Dimension, Multiclass, Split-ratio, Balance, and Data-type as shown in Table 1. The length of the dataset refers to the sequence length of the time series. The shortest dataset is Electric Devices, which has a length of 96, while the longest is Ethanol Concentration, which has a length of 1751. Strength refers to the number of total samples in a dataset. Electric devices have the highest strength, followed by Pen Digits. Self-regulation SCP1 has the lowest strength with 561 samples. Dimensionality is the number of input feature; the higher the dimension, the more complex. We similarly perform categorization depending on the number of output classes, referred to as multiclass. Electric Devices and Pen Digits are multiclass datasets, while "Star Light Curves" and "Ethanol Concentration" are not. This attribute is crucial for classification tasks. All of our datasets are uni-labelled; the output is a single dimension.

For the benchmark to be consistent with the provided dataset to understand the impacts of the model, we used the original train-test split provided. We compute the split ratio, that is, the ratio between the train test splits. Most datasets have around one; if it is greater than one, it has too few testing samples compared to training, and less than 1 indicates few training samples compared to testing. For the balance of the class distribution, we again have two categories, balanced or unbalanced, which are assigned depending on the class ratio.

Table 1. Dataset characteristics

Dataset	Length	Strength	Dimension	Multiclass	Split-ratio	Balance	Data-type
Star Light Curves	1024	9236	1	3	0.12	Unbalanced	Sensor
Electric Devices	96	16637	1	7	1.15	Unbalanced	Device
ECG 5000	140	5000	1	4	0.12	Unbalanced	ECG
Ethanol Concentration	1751	1004	1	5	1	Balanced	Spectro
Pen Digits	8	10992	2	10	2.14	Balanced	Motion
Self Regulation SCP1	896	561	6	2	1	Balanced	EEG
Racket Sports	30	303	6	4	1	Balanced	HAR
NATOPS	51	360	24	6	1	Balanced	HAR

3.2 Deep learning models

In this section, we examine a range of deep learning models for time series classification utilized in the study. The models are generally categorized into three types: basic simple models, extensive deep models, and hybrid models integrating both CNN and RNN structures in their design.

LSTM network [8] - Effective for tasks like time forecasting and anomaly detection, it can learn long-term dependencies through its memory structure. The Bi-LSTM processes sequences in both forward and backward directions.

FCN network [14] - Utilizes 1D CNN blocks for time series, featuring layers that upsample and downsample data. Average pooling reduces output dimensions before feeding into a linear layer to obtain class probabilities as output. We have also tested the same architecture using 2dCNN, which is known as **Convent2d**.

Resnet Network [7] - Resnet blocks are used to build the network. The main concept introduced is residual connections in the block, which help in the convergence and stabilization of the network. These blocks are used to upsample the input data and then apply average adaptive pooling, followed by a linear layer.

InceptionTime [9] - It is an ensemble of six Inception modules [20] with different randomly initialized weight values. The inception module is composed of CNN layers with varying filter lengths. This allows the neural network to extract relevant features from long and short-time series.

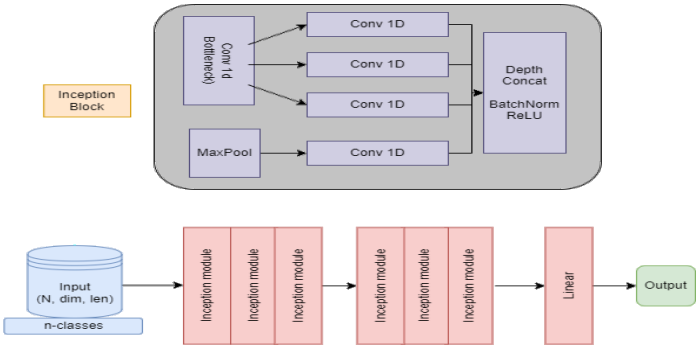

Fig. 1. InceptionTime architecture

XceptionTime [17] - Its architecture integrates depthwise separable CNN with different filter sizes, adaptive average pooling, and a non-linear normalization technique. They are concatenated in series to capture temporal and spatial information without manually extracting features.

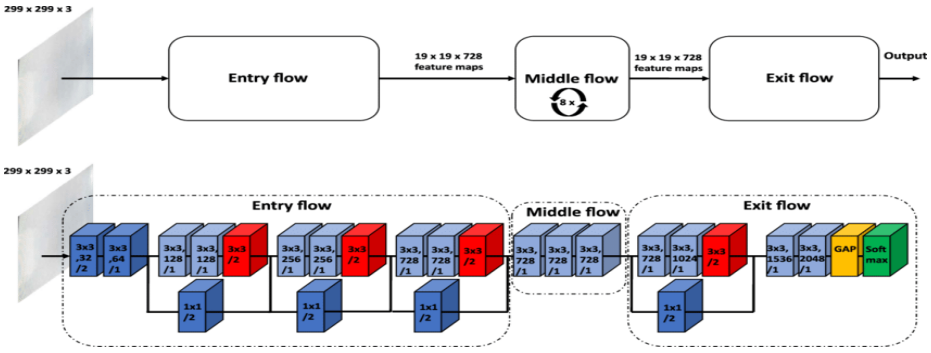

Fig. 2. XceptionTime architecture

dCNN [2] short for dimension-wise Convolutional Neural Network was introduced recently to overcome the limitations of traditional CNNs in handling multivariate time series data. The architecture of dCNN is similar to that of a traditional CNN model, the difference lies in the input to the model, which is transformed into a cube, in which each row contains a combination of all di-

mensions. This helps the model learn temporal and dimensional discriminant information. We have also used dResnet similar to dCNN as mentioned in the original paper [2].

XCM [6] - Designed for multivariate time series, it merges 2D and 1D CNNs in a parallel architecture with upsampling and downsampling layers. Their output is concatenated in the first dimension and passed through an upsampling CNN layer. A linear layer at the end follows a pooling layer to get the output.

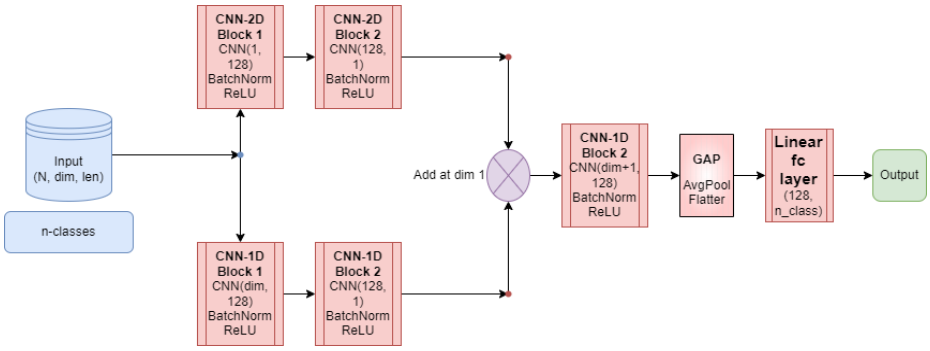

Fig. 3. XCM model architecture

LSTM-FCN [12] - Combines LSTM and FCN in parallel, allowing for concatenation of results and the potential addition of an attention mechanism to weigh the importance of sequence components.

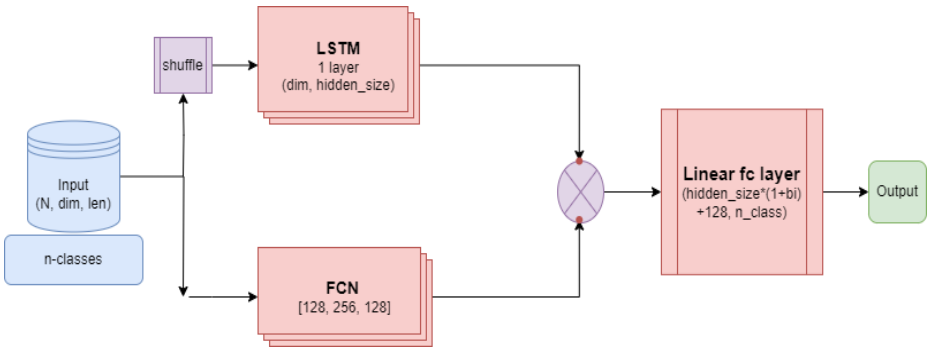

Fig. 4. LSTM-FCN model architecture

We have introduced a smaller hybrid architecture LSTM-2dCNN similar to the LSTM-FCN. **LSTM-2dCNN** consists of a parallel network of LSTM and 2DCNN layers. The output of the 2DCNN is reshaped to allow for the concatena-

tion. In this case, after the concatenation of the outputs of the parallel network, the result is passed through the dropout and linear layer.

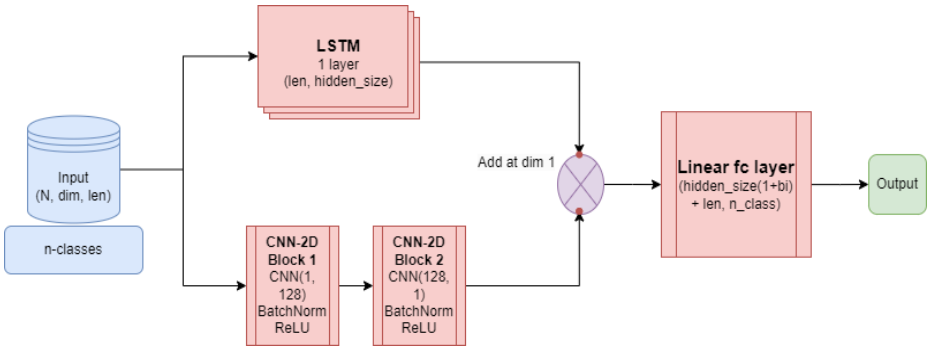

Fig. 5. LSTM-2DCNN model architecture

3.3 Training scenarios

In this study, we have experimented with four different training algorithms. This section describes the different training scenarios.

1. First, we use the LR finder to find the correct range and then use the AdaptiveCycle LR scheduler, which we have introduced in the paper.
2. First, using LR finder to find the correct range and then using OneCycle LR scheduler.
3. Using LR finder to find the optimal LR (the LR resulting in minimum loss in the loss curve) to train the model and then use static LR for training.
4. Using a static LR to train the neural network, which has been set to 1e-5.

3.4 Evaluation metrics

The different evaluation metrics used for the benchmarking model and the five training scenarios performances are explained in this subsection. These will help us to understand the effect of various characteristics on performance.

Accuracy is a percentage of correctly classified inputs over the total number of inputs in the testing dataset. The higher the accuracy, the better is modal performance. However, in the case of an imbalanced dataset, it might be misleading to use accuracy, as it cannot detect the model bias due to the imbalanced dataset.

F1score is the harmonic mean of precision and recall. It is useful, especially in the case of imbalanced data, as we also take into account the false positives and false negatives to calculate the precision and recall. Similar to accuracy, the higher the F1 score, the better the model performance.

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}} \tag{1}$$

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}} \tag{2}$$

$$F1 \text{ Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{3}$$

Convergence speed, in our case, has been quantified as the amount of time a model requires to converge, that is, to have an optimal performance by stable loss during training. It is one of the most important metrics for our study. Faster convergence means efficient training, which helps save computational resources.

3.5 Experimentation setup

We have used cross-entropy loss for backpropagation during training and Adam as an adaptive optimizer for all four scenarios mentioned previously. We have used Pytorch to build all our functions and the customer schedulers instead of fastai, as it provides more transparency, greater clarity, and control over the training process. An L4 GPU of 16GB from Nvidia has been used for training and testing these models. We have used PyTorch dataloaders and normalized the data. Normalizing the data provides scaled uniformity and helps avoid inconsistencies caused by batch-level statistics during training.

4 AdaptiveCycle Methodology

In this section, we go through the main process of training the deep learning model with a detailed explanation of the LR finder, OneCycle LR and AdaptiveCycle LR schedulers. Parameters from the dataset, such as sequence length, number of classes and dimension, are extracted and used for model creation. The loss criteria and optimization are initialized.

LR Finder is used to calculate the optimal range of LR. This range is calculated by increasing the LR exponential after each batch iteration. Usually, the LR finder fits for just one epoch, but this might lead to unstable results when the dataset is small, as it might not provide enough iterations to form a stable curve. We have modified the existing LR finder to be used over more epochs. The user can select this by setting the multi-epoch value as True or False. Figure 6 shows the difference between with and without multiple epochs for the ECG5000 dataset. In this case, it is more beneficial to use multi-epochs. In the Results section, we discuss the scenarios depending on the dataset in which setting the multi-epochs as True is more beneficial.

Another important hyperparameter affecting the training and model optimization is the weight decay (Wd) we set for the optimizers. Figure 7 shows the LR finder loss curve initialized with different Wd, and we can see that it does have an effect. Usually, lower Wd works well for simple models, as there is less

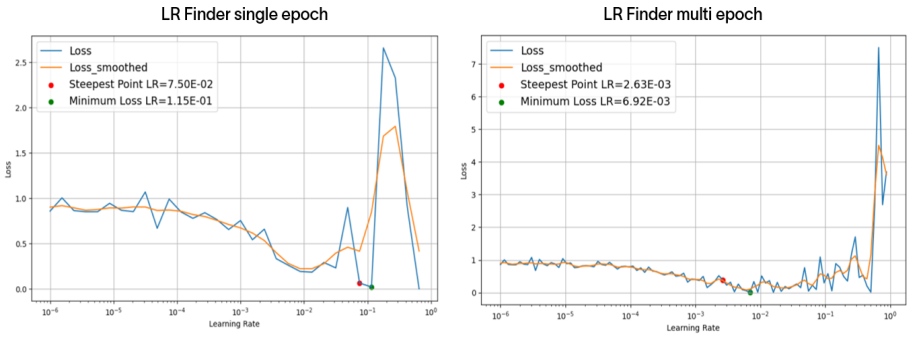

Fig. 6. LR finder results for ECG using LSTM-FCN model

chance of overfitting. At the same time, for complex architectures, a higher Wd can help regularise the model by preventing heavy reliance on specific features. We suggest performing a grid search of Wd using an LR finder. It will efficiently capture the effects of the Wd and help pick the best one according to the loss curves. The ideal Wd should have a smooth loss curve with less divergence. We provide analysis on Wd in the Results section.

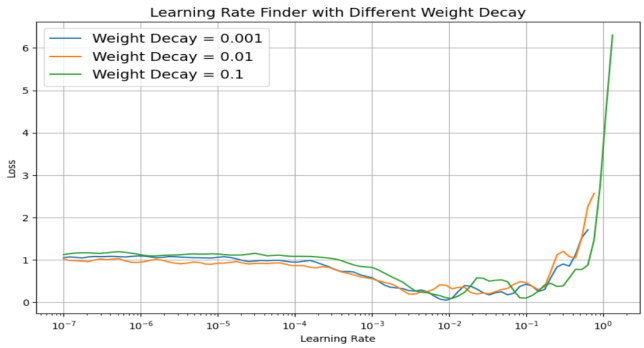

Fig. 7. Affect of weight decay on loss for ECG using LSTM-FCN model

Before going into detail about the AdaptiveCycle LR, it is important to understand the OneCycle LR and why there is a need to upgrade it to a more customizable and controllable LR scheduler. OneCycle can be explained by dividing the training into four phases depending on the LR curve during training. We start with a warm-up phase where the LR starts low at the set lr_min and gradually increases. This helps the model stabilize and avoid large weight updates at the start of training. Then, the high LR phase, where it is around the set lr_max value. It helps train faster and explore the lost landscape to avoid local minima. In the decay phase, the LR decreases and returns to lr_min, where

the model is fine-tuned. Then, in the final phase, the LR is reduced further for just a few iterations, allowing the model to settle well into its optimal values.

The momentum follows a cycle inverse to the LR if set to true. It starts from high, goes to a minimum, and then back to the original high value. However, during the final phase of LR, the momentum also decreased slightly to avoid a prominent change in model weight in the final phase. We can set the percentage of cycle used for the increasing LR, which is usually set to 30 or 50.

AdaptiveCycle LR is a custome scheduler that is a combination of OneCycleLR with the concept of reducing the LR value when there is no improvement in the validation and training accuracy. This allows the validation performance to have an impact on the LR adjustments. This increases our control over the LR scheduler by offering to flexibly set the parameters controlling conditions to reduce LR and reinitializing the OneCycle LR with updated parameter values. The reduction of LR depending on validation loss helps stabilize the learning, especially during the warm-up and high LR phases, preventing overfitting and improving generalization. It helps in better fine-tuning the model. The algorithm of the AdaptiveCycle LR helps us understand the concept better 1.

Algorithm 1 AdaptiveCycle Learning Rate Scheduler

Require: initial minimum learning rate lr_{min}, maximum learning rate lr_{max}, final minimum learning rate lr_{fmin}, cycle length T, epochs E, patience p, factor f
  Initialize OneCycleLR($lr_{min}, lr_{max}, lr_{fmin}, T, E$)
  least_validation_loss = ∞
  for epoch = 1 to N **do**
    Start OneCycleLR: $lr \leftarrow lr_{min}$
    $epochs\_patient \leftarrow 0$
    if epochs_patient $> p$ **then**
      When epoch $< 1, current\_loss \leftarrow$ Validation Loss()
      if $current\_loss < least\_validation\_loss$ **then**
        $least\_validation\_loss \leftarrow current\_loss$
      else
        $epochs\_patient \leftarrow epochs\_patient + 1$
      end if
      for Iteration per batch **do**
        $lr_{new} \leftarrow$ OneCycleLR($lr_{min}, lr_{max}, lr_{fmin}, T, E$)
      end for
    else if epochs_patient $> p$ **then**
      $lr\_old = lr\_current$
      Apply Reduction policy $lr\_new = lr\_current * f$
      Reinitialise the OneCycleLR
      OneCycleLR($lr_{min} = (lr\_new)0.5, lr_{max} = lr\_new, lr_{fmin}, T, E$)
    end if
  end for

Figure 8 shows the AdaptiveCycle LR applied in the dataset of Electric devices and Racket sports when trained using the LSTM-FCN model. Observing

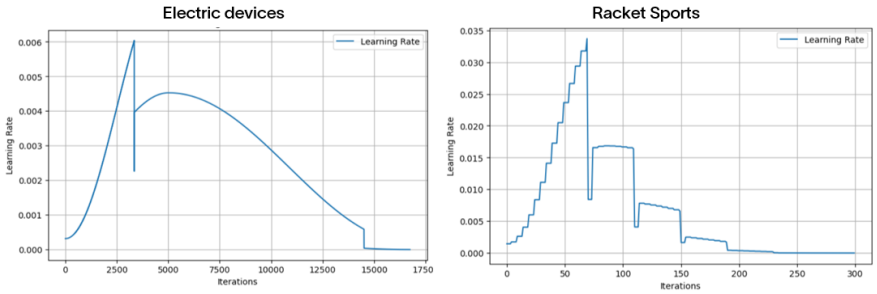

Fig. 8. AdaptiveCycle LR curve when training LSTM-FCN model

the LR curves helps us better understand the algorithm. For the electric device dataset, we have set the cycle length as default, which means that after each batch iteration, the algorithm changes the LR value, resulting in a smooth curve. For the Racket Sports dataset, we have set the cycle length to $T_{original}/5$. Thus, the LR update now occurs every 5 iterations, and can be observed in the image as steps. When a plateau in validation loss is reached, the LR reduction occurs and the OneCycle is reset.

5 Benchmark Results

In this section, we provide a discussion on the results of the experiments conducted with different training methods and on the evaluation metrics mentioned in the section on benchmark setup. We also make inferences with respect to the dataset features and try to generalize the best model in each case.

The accuracy plot in Figure 9 consists of 2 subplots, the first representing a box plot of average accuracy across the eleven models used per dataset and the second representing the box plot of model accuracies across the eight datasets. We will first focus on the training methods; both AdaptiveCycle LR and OneCycle LR have provided better accuracy when compared to training without schedulers. We can see that the variance for AdaptiveCycle LR is the least across models and datasets.

Figure 10 represents the training and validation loss curve when using the two schedulers; it can be seen that the AdaptiveCycle LR proposed provides greater stability over the training compared to OneCycle LR. In the second half of the training cycle, due to the decreasing LR, both methods lead to stability, but the gap between the train and validation loss is greater at the end in OneCycle. We also found that the variation across different initial seeds is more when using OneCycle LR vs the AdaptiveCycle.

Similar to the accuracy plot, we also have the F1 score plot displayed in Figure 11. The F1 score plot is important to observe when the training dataset has an imbalanced class distribution. We have a skewed distribution of classes

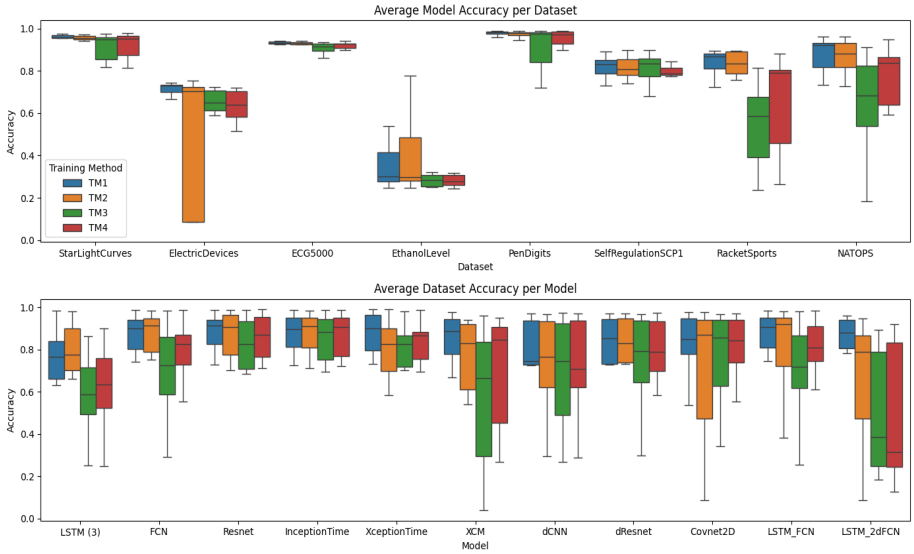

Fig. 9. Accuracy plot

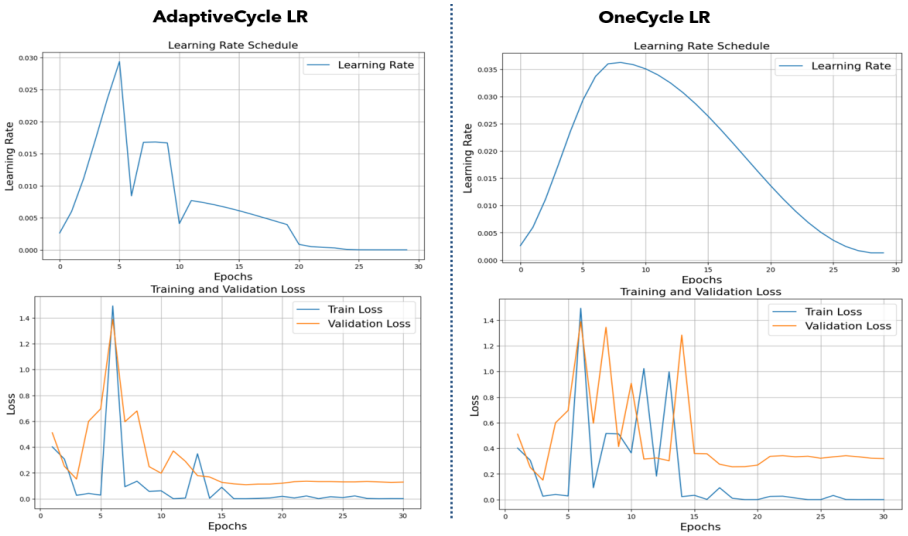

Fig. 10. Loss curves for Racket sports with LSTM-FCN model

in the Starlight Curves, Electric devices, and ECG datasets. Among them, the F1 score does not decrease compared with accuracy for Starlight Curves and Electric devices; this might be due to the larger dataset, indicating that the model can still learn the class attributes when enough samples are available.

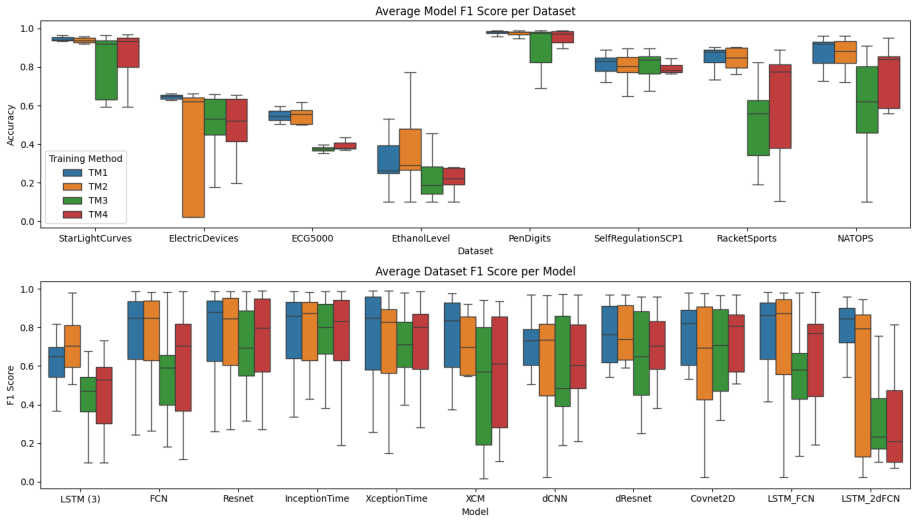

Fig. 11. F1 score plot

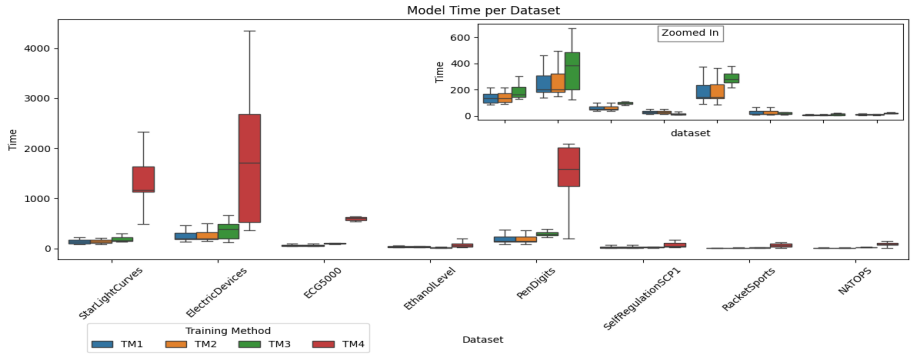

Fig. 12. Time per dataset

This is not the case for ECG5000, as we have less than 20 samples for three of the five classes, leading to a performance decrease. It can also be observed that the performance of the LSTM model is lower when compared to the accuracy in Figure 9, indicating that the model has not been generalized well across the different classes.

The model scores per dataset plot allows us to compare the different training methods, while the dataset scores across models will enable us to categorize the model's performance. After comparing the accuracy and the f1 score plots, we can conclude that the Resnet and InceptionTime are stable models converging well across the different training methods. Both models comprise numerous CNNs

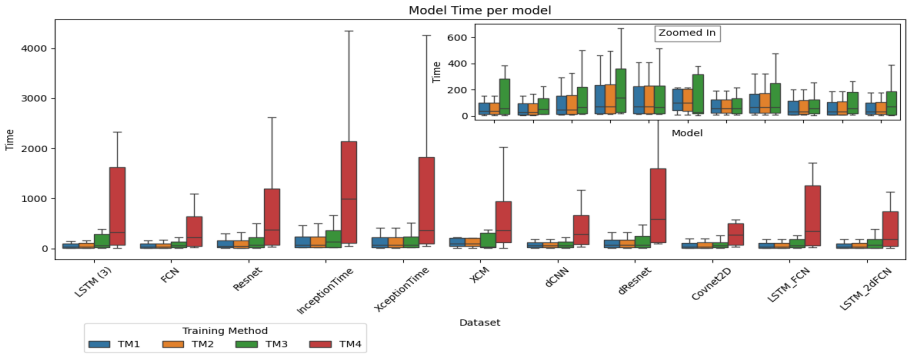

Fig. 13. Time per model

and feature deep convolutional architectures. The LSTM model is the one with the worst performance.

We have observed that the OneCycle needs more fine-tuning when selling the lr-max and lr-min. Incorrect settings may result in the model being unable to reach the optimal solution, as observed for the Electric devices dataset. Especially when the lr-max is a bit higher, it leads to an unstable model. As OneCycle follows a predefined pattern of a fixed increase schedule followed by a decreasing LR, it is inflexible. The AdaptiveOneCycle can address this issue as it changes dynamically and is controlled by the validation loss; it helps prevent scenarios of large oscillation during training due to high LR.

For evaluating the performance against computational time, we will be referring to the two Figures 12 and 13. Schedulers would be less time-consuming than traditional learning with a low static learning rate. Each figure contains a zoomed plot; they have the same consumption time since we run the schedulers for about 50 epochs. Among the datasets, Electric devices have taken the most time, as they have the largest data samples for training. Resulting in more batches which leads to more iterations to be performed per epoch. When comparing across models, it can be seen that the LSTM, FCN, Convnet2D, LSTM-FCN, and LSTM-2dFCN are among the faster models. Inception Time and Xception Time require much more as they are deeper architectures with more CNN layers.

The parameter size of XCM, Convnet2D, LSTM-FCN, and LSTm-2dFCN are heavily dependent on the sequence length of the dataset, while the others are more dependent on the dimension of the input data. However, the parameter size has a negligible effect on the model's computational time and accuracy scores. This is due to the use of GPUs, designed to handle large-scale parallel computations efficiently. GPUs' high memory bandwidth and parallel processing capabilities allow them to manage large parameter sizes without significant performance degradation.

6 Conclusion

Using an LR scheduler is the way to train deep learning models efficiently. We can obtain better and faster convergence without affecting the model performance. Both the OneCycle and AdaptiveCycle use the same initial strategy of warmup of LR for exploration, followed by fine-tuning the model during the LR descent. The OneCycle is easy to use, with its implementation being available in many libraries such as Pytorch and fastai. Its only disadvantage is that during the high LR, the model oscillates a lot, which may, in some cases, lead to a non-optimal modal solution, as observed in the paper.

AdaptiveCycle, proposed by us, helps resolve most of the disadvantages of using just OneCycle. It gives more fine-grained control over the learning rate adjustments based on validation performance while still keeping the advantages of OneCycle with faster convergence. It provides improved stability and performance by decreasing the LR in cases when the oscillation of loss occurs, especially in handling large, complex datasets. We hope the results of our benchmark study will lead to more people using schedulers, as it will allow the research time to be efficiently spent on data and model architecture rather than training models, while also saving resources. We would also release open-source code for our project, allowing people to use the customer scheduler.

Acknowledgments. We would like to acknowledge the UCR and UCE Time Series Classification Archive [4] for providing the datasets used in this study.

References

1. Akl, A., El-Henawy, I., Salah, A., Li, K.: Optimizing deep neural networks hyperparameter positions and values. Journal of Intelligent & Fuzzy Systems **37**(5), 6665–6681 (2019)
2. Boniol, P., Meftah, M., Remy, E., Palpanas, T.: dcam: Dimension-wise class activation map for explaining multivariate data series classification. In: Proceedings of the 2022 International Conference on Management of Data. p. 1175–1189. SIGMOD/PODS '22, ACM (Jun 2022). https://doi.org/10.1145/3514221.3526183, http://dx.doi.org/10.1145/3514221.3526183
3. Darke, P., Missier, P., Bacardit, J.: "benchmark time series data sets for PyTorch - the torchtime package" (July 2022). https://doi.org/10.48550/arXiv.2207.12503, https://doi.org/10.48550/arXiv.2207.12503
4. Dau, H.A., Keogh, E., Kamgar, K., Yeh, C.C.M., Zhu, Y., Gharghabi, S., Ratanamahatana, C.A., Yanping, Hu, B., Begum, N., Bagnall, A., Mueen, A., Batista, G., Hexagon-ML: The ucr time series classification archive (October 2018), https://www.cs.ucr.edu/eamonn/time_series_data_2018
5. Duchi, J., Hazan, E., Singer, Y.: Adaptive subgradient methods for online learning and stochastic optimization. J. Mach. Learn. Res. **12**(null), 2121–2159 (Jul 2011)
6. Fauvel, K., Lin, T., Masson, V., Fromont, É., Termier, A.: Xcm: An explainable convolutional neural network for multivariate time series classification. Mathematics **9**(23), 3137 (Dec 2021). https://doi.org/10.3390/math9233137, http://dx.doi.org/10.3390/math9233137

7. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. CoRR **abs/1512.03385** (2015), http://arxiv.org/abs/1512.03385
8. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural Computation **9**(8), 1735–1780 (1997). https://doi.org/10.1162/neco.1997.9.8.1735
9. Ismail Fawaz, H., Lucas, B., Forestier, G., Pelletier, C., Schmidt, D.F., Weber, J., Webb, G.I., Idoumghar, L., Muller, P.A., Petitjean, F.: Inceptiontime: Finding alexnet for time series classification. Data Mining and Knowledge Discovery **34**(6), 1936–1962 (Sep 2020). https://doi.org/10.1007/s10618-020-00710-y, http://dx.doi.org/10.1007/s10618-020-00710-y
10. Johnson, O.V., XinYing, C., Johnson, O.E., Khaw, K.W., Lee, M.H.: Learning rate schedules and optimizers, a game changer for deep neural networks. In: Saeed, F., Mohammed, F., Fazea, Y. (eds.) Advances in Intelligent Computing Techniques and Applications. pp. 327–340. Springer Nature Switzerland, Cham (2024)
11. Kadhim, Z.S., Abdullah, H.S., Ghathwan, K.I.: Artificial neural network hyperparameters optimization: A survey. Int. J. Online Biomed. Eng. **18**(15), 59–87 (2022)
12. Karim, F., Majumdar, S., Darabi, H., Chen, S.: Lstm fully convolutional networks for time series classification. IEEE Access **6**, 1662–1669 (2018). https://doi.org/10.1109/access.2017.2779939, http://dx.doi.org/10.1109/ACCESS.2017.2779939
13. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization (2017), https://arxiv.org/abs/1412.6980
14. Long, J., Shelhamer, E., Darrell, T.: Fully convolutional networks for semantic segmentation. In: 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). pp. 3431–3440 (2015). https://doi.org/10.1109/CVPR.2015.7298965
15. Napoli Spatafora, M.A., Ortis, A., Battiato, S.: Glr: Gradient-based learning rate scheduler. In: Foresti, G.L., Fusiello, A., Hancock, E. (eds.) Image Analysis and Processing – ICIAP 2023. pp. 269–281. Springer Nature Switzerland, Cham (2023)
16. Pandey, M., Fernandez, M., Gentile, F., Isayev, O., Tropsha, A., Stern, A.C., Cherkasov, A.: The transformational role of gpu computing and deep learning in drug discovery. Nature Machine Intelligence **4**(3), 211–221 (2022)
17. Rahimian, E., Zabihi, S., Atashzar, S.F., Asif, A., Mohammadi, A.: Xceptiontime: Independent time-window xceptiontime architecture for hand gesture classification. In: ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). pp. 1304–1308 (2020). https://doi.org/10.1109/ICASSP40776.2020.9054586
18. Smith, L.N.: Cyclical learning rates for training neural networks (2017), https://arxiv.org/abs/1506.01186
19. Smith, L.N., Topin, N.: Super-convergence: Very fast training of neural networks using large learning rates (2018)
20. Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., Rabinovich, A.: Going deeper with convolutions. In: 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). pp. 1–9 (2015). https://doi.org/10.1109/CVPR.2015.7298594
21. Wang, W., Zhang, M., Chen, G., Jagadish, H., Ooi, B.C., Tan, K.L.: Database meets deep learning: Challenges and opportunities. ACM Sigmod Record **45**(2), 17–22 (2016)
22. Xu, Z., Dai, A.M., Kemp, J., Metz, L.: Learning an adaptive learning rate schedule (2019), https://arxiv.org/abs/1909.09712