THIS IS NO LONGER IN USE. ONLY EDIT THE LATEX FILE in our build.ge repo

ARC – Autonomous learning rate scheduler

ARC – Autonomous learning rate controller

To Raise or Not To Raise: The Great Learning Rate Question

(any better name is welcome)

**Abstract**

the source code will be made available.

**1. Introduction**

Learning rate is one of the most important hyperparameters in deep learning training, a parameter everyone interacts with for all tasks. In order to ensure model performance and convergence speed, learning rate needs to be carefully chosen. Otherwise, a big learning rate will cause divergence and a small learning rate will lead to inefficient training. Furthermore, as complexity of training evolves over time, besides a carefully chosen learning rate, learning rate scheduling have also gained favor with many state-of-the-art AI applications[][][][]. Learning rate scheduling provides finer control of learning rates by allowing different learning rates to be used throughout the training. However, the extra flexibility comes with a cost - more parameters to tune. Given the tradeoff, there are broadly two ways of approaching learning rate scheduling by AI community.

Experts with sufficient compute resources tend to hand-craft their own learning rate schedules, because a well-customized learning rate schedule can often result in improving current state-of-the-art, putting them on top of leaderboard. For example, entries in Standford Dawn Benchmark leaderboard [] are known for using carefully tuned learning rate schedules to achieve the world-record convergence speed. However, such learning rate schedules suffer great drawbacks too. First, these learning rate schedules are often specifically tailored to the exact configuration (architecture, dataset, optimizer) such that they do not generalize to other tasks. Moreover, hand-tuning a good learning rate schedule is more than trial and error, as it also involves following certain heuristics based on observations. As a result, building a well-customized learning rate schedule often requires great expertise and significant amount of compute resources.

In contrast, others favor existing task-independent learning rate schedules, as they often provide decent performance gain with less tuning efforts. Some popular choices are cyclic cosine decay[], linear warmup[], exponential decay[]. While these learning rate schedules can be used across different tasks, on the flip side, however, being agnostic of task also means ignoring key training characteristics. As a result, these schedules do not guarantee performance improvement. On top of that, many of such learning rate schedules still require significant amount of tuning to work well. For example, in cyclic cosine decay[], parameters such as l\_max, l\_min, T\_0 and T\_multi still require further tuning to function properly.

Recent Advancements in AutoML on Architectual search[nas][enas][darts] and update rule search[] have proved that it is possible to create an automated system that performs equivalently well as human experts on deep learning design. Those successes have inspired us to tackle the learning rate scheduling problem. Considering above problems and challenges faced by different groups of AI practitioners, we aim to create a system that learns how to change learning rate effectively.

For that, we propose ARC – an autonomous and generalizable learning rate scheduling system that can change learning rate intelligently based on multiple training signals. As we will illustrate in section 2 and 3, ARC can effectively overcome the challenges faced by both groups of people. To experts, ARC has encoded the knowledge and experience of learning rate changing in a generalizable way, such that the model can be easily used on different tasks. In the meantime, ARC is also what non-experts would expect an autonomous learning rate scheduler should be: Instead of relying on predefined functions that ignore training characteristics, ARC monitors the training activity for specific task and make corresponding change on learning rate automatically for a better results and faster convergence.

Our contribution in this work includes:

1. The overall methodology of the autonomous learning rate system, including learning problem framing, data collection design and ground truth correction is novel and can provide inspirations to later study.
2. The ARC system is deployed on unseen learning problems, then compared against popular learning rate schedules mentioned above in terms of model performance and convergence speed across multiple computer vision and language tasks.
3. Failure modes and limitations of ARC are demonstrated together with future directions of improvement. Several surprising phenomena about the relation between learning rate and overfitting are also observed.

**1.5 Challenges and Constraints**

Our goal is to design intelligent system that can change learning rate automatically from available inputs. Before delving into the methodology, it is worthwhile to mention the challenges as the rest of the sections are focused on addressing them.

a) Given any point in the training process, a “successful” learning rate is subjective. This is because a good learning rate is measured by multiple indicators that may not be consistent with each other. Such indicators include training loss, validation loss and task-specific metrics like accuracy, dice. A lower training loss certainly does not equate to lower validation loss and lower validation loss doesn't necessarily lead to better task-specific metric either.

b) Associating model performance to specific a learning rate on a step basis is challenging. This is due to the fact that the training trajectory is a cumulative effect of all previous learning rates, such that one cannot simply infer a direct causal relation between a specific learning rate and its performance on a step basis.

c) The randomness during the training makes it harder to compare two learning rates. Some common sources of randomness are dataset shuffling, data augmentation, random network layers such as dropout. As a result, when comparing two learning rates, we need to make sure the learning rates are different enough such that their impact in the training can overcome the randomness.

d) The available history for different learning tasks varies in variety and scale, making it challenging to generalize to new tasks. Different deep learning training generates different histories such as of training loss, validation loss and task-specific metrics. Among them, training loss and validation are the only available information that is task-independent. However, the loss measurement itself can vary greatly across different tasks. For example, categorical cross entropy for 1000-class classification usually lies within the range of (0, 10], but a pixel-level cross entropy loss for segmentation can easily reach a scale of several thousand. Moreover, one can easily design custom loss function with negative loss values.

e) Learning Rate control system must have small footprint. The purpose of having an automated learning rate control system is to guide the deep learning training for faster convergence and better results. However, it would defeat its purpose if such learning rate control model’s footprint is large enough to affect training speed and memory consumption.

**2.Methods**

**2.1 Framing the learning rate change as learning problem.**

For the learning task, we frame it as supervised learning problem – predicting next learning rate given available training history. Because of challenge (b), the model needs to observe the consequence of a specific learning rate for long enough to form a clear association between learning rate and its performance, therefore, we want the model to only change learning rate on an epoch basis.

For the inputs of the model, according to challenge (d), any task-specific metrics cannot be used because our intent is to generalize to new task. Any model parameter or gradient related property would require significant amount of footprint and therefore cannot be used either due to challenge (e). As a result, for the model inputs, we found the history of training loss, validation loss and previous learning rates are the least demanding yet informative inputs we can use.

For the outputs of the model, as mentioned in challenge (c) that a better learning rate is only observable on a coarse scale, therefore, we frame the learning rate prediction as classification problem for 3 discreet classes. Each class represents a multiplication factor on previous learning rate: [1.618, 1.0, 0.618]. In other words, given inputs, we want the model to predict following 3 actions on current learning rate: 1) increase by factor of 1.618, 2) do nothing, 3) decrease by factor of 0.618. There is no theoretical reason in choices of these 3 numbers except that the combined effect of one increase and one decrease can approximately lead to the original value.

**2.2 Collecting training history and ground truth from multiple training tasks**

Now that we have defined the learning problem, what we need next is generating enough inputs-ground truth pairs from real deep learning training. We designed the following workflow to collect our data:

1) train n epochs with learning rate lr, then save the current state as checkpoint C

2) reload C, train for another n epochs with 1.618 lr, calculate validation loss at the end.

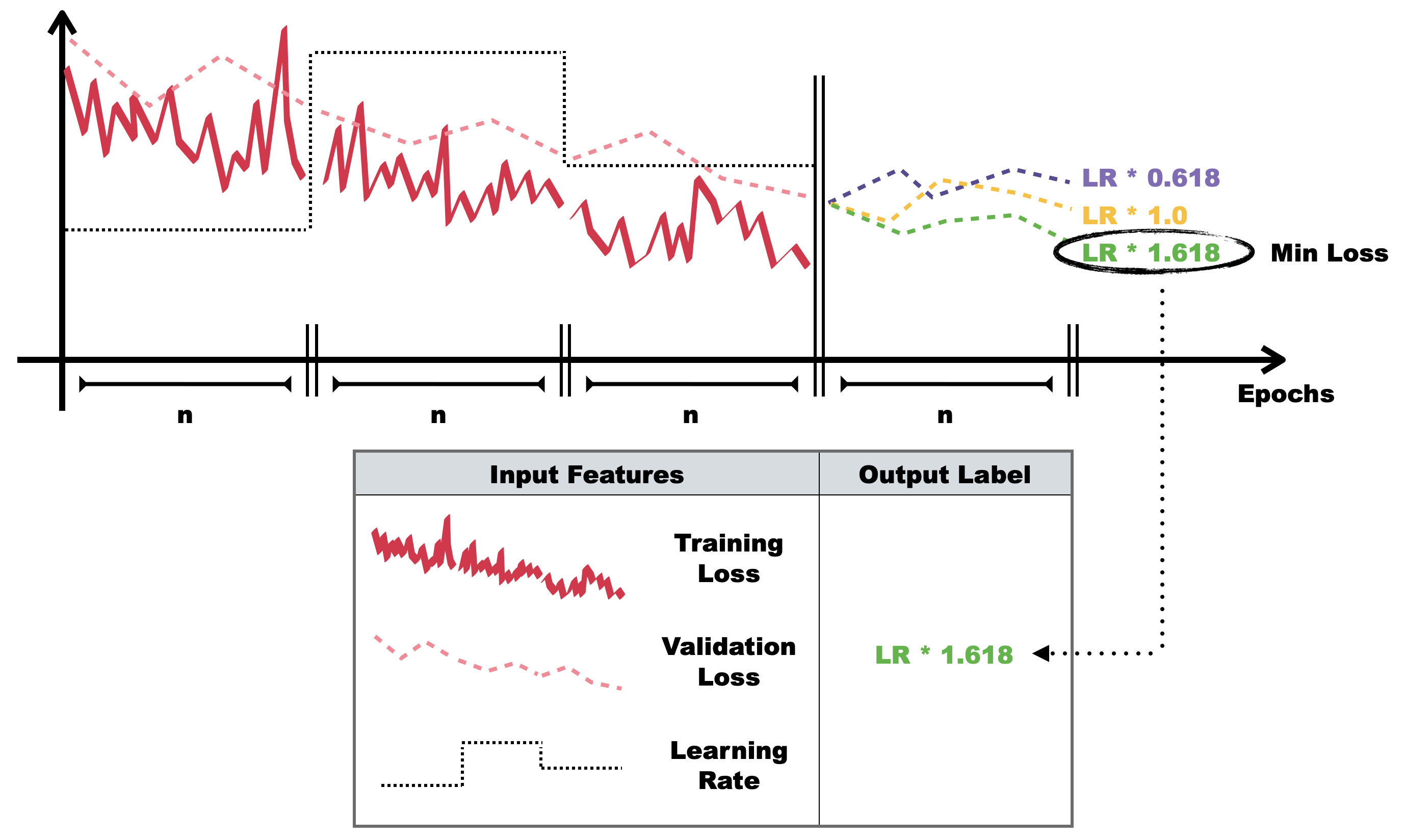
3) reload C, train for another n epochs with lr, calculate validation loss at the end.

4) reload C, train for another n epochs with 0.618 lr, calculate validation loss at the end.

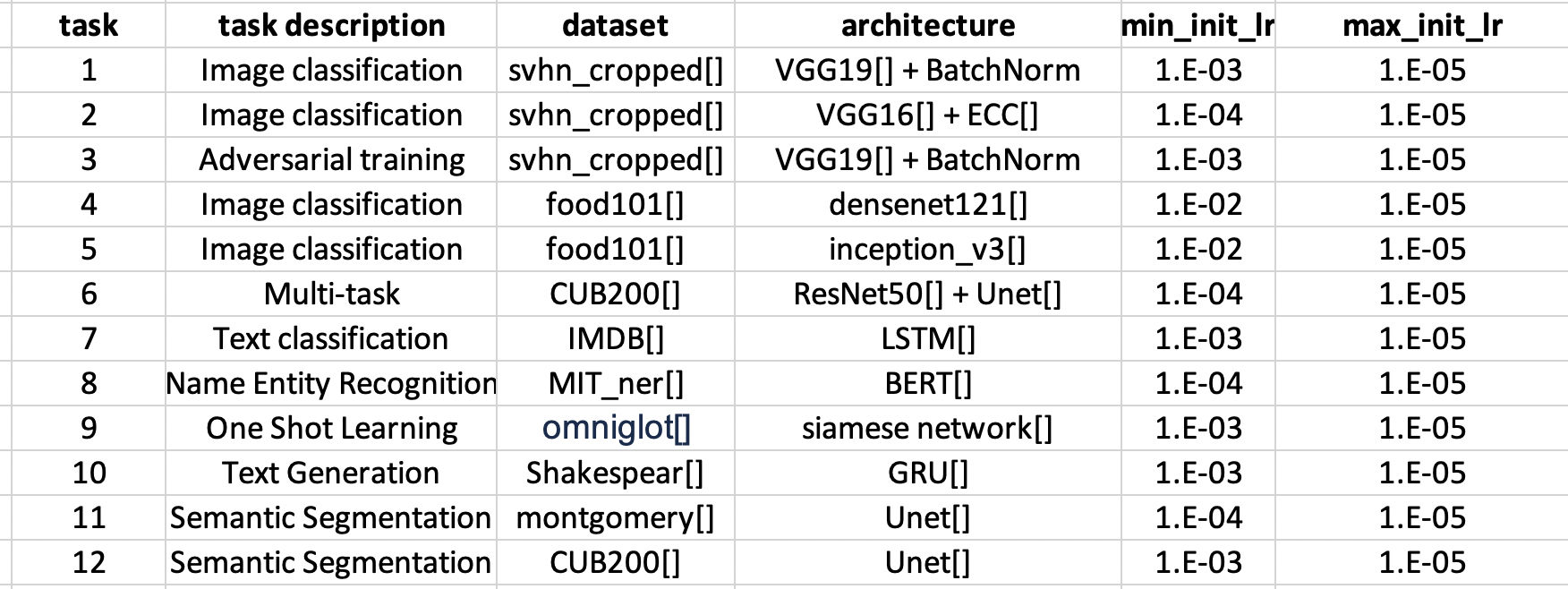
5) reload C, among (2),(3) and (4), eliminate the option with the highest validation loss, next randomly select among the rest two and make the corresponding change to lr

By executing (a) – (e), we can create one inputs – ground truth pair; Inputs are training loss, validation loss and learning rate generated during (a) plus the inputs data from previous 2n epochs. For ground truth, given challenge (a), we assume a better learning rate is a learning rate that leads to a lower validation loss after n epochs. Therefore, the option with lowest validation loss is chosen as ground truth label. The entire workflow is depicted in Figure X. (comments: we need to mark the starting point in figure)

Now that we have generated one sample point, next we repeat (1) to (5) until training finishes, therefore, if the total number of epochs is N, we can get N/n sample points. In 5) we eliminate the worst loss and randomly choose among the best 2 options because we want to capture as many scenarios as possible without causing the loss to diverge.



To ensure the data covers a wide variety of tasks, we gathered 12 different computer vision and language tasks and each task has a unique configuration (datasets, architectures, optimizers, initial learning rate etc.) as shown in Table X. Each task is trained on average 30 times with n randomly ranging between [1, 10], total number of epochs to be 10n, optimizers are randomly selected from [Adam, SGD, RMSprop]. In the end, around 4500 samples are collected in total.



**2.3 Correcting ground truth**

As stated in challenge (c), the ground truth on learning rate may be accompanied by multiple sources of randomness. For example, if the validation loss that is associated with 3 actions is [0.113, 0.112, 0.111], we may not be very confident claiming decreasing learning rate is the best action due to the randomness.

Luckily, there is one more datapoint we can use to reduce uncertainty. Assuming step (5) of Section 2.2, we choose option (4) - decrease learning rate, then the next round of (1) is doing exactly the same as last round of (4) because they both start from the same checkpoint and train with the same learning rate. As a result, we can collect one more validation loss. Assuming the validation loss from the end of next round of (1) is 0.109. Then our validation loss for the 3 actions becomes [0.113, 0.112, (0.111, 0.109)].

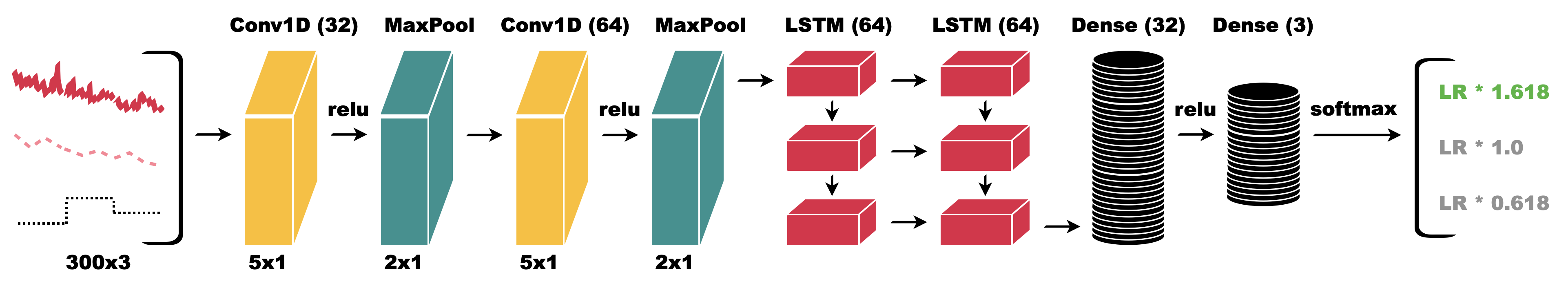
Now that we have two loss candidates for option (4), we will sort the options based on validation loss using each candidate, if the sorting result does not change, then the ground truth is “confirmed”. For example, the sorting result of [0.113, 0.112, 0.111] and [0.113, 0.112, 0.109] is same, therefore, we can confirm option (4) - decreasing learning rate is indeed the ground truth.

On the other hand, if the sorting results are inconsistent, then it is an indicator of uncertainty in ground truth. Therefore, we are going to follow the philosophy: if we are not sure about changing learning rate, then better not to change it. As a result, we will manually assign ground truth to be option (3) - do nothing.

**2.4 Building the model**

For preprocessing, to standardize loss with various scales as mentioned in challenge (d), we apply z-score on all training losses and validation losses. For data points that are collected during early training phase, their inputs data will be missing on previous 2n or 3n epochs. We will then pad the missing data with zero after applying zscore. We normalize all learning rates by dividing by its first value. At last, training loss, validation loss and learning rate are all resized to size 300 using bilinear interpolation then concatenated in new dimension, making the final shape of inputs to be (batch, 300, 3).

The model architecture is shown in Figure X, there are 3 essential components: feature extractor, LSTM[] and dense classifier. Feature extractors include 2 layers of 1D convolution, they can play a role as a sophisticated 1D sliding window feature extractor. The LSTM comprised of two layers of stacked LSTM units. At last, two densely connected classifiers are used for final prediction. Considering the limitation on footprint mentioned in challenge (e), the parameters of each layer is carefully chosen such that the overall number of trainable parameters is less than 80k. The model is trained for 300 epochs using 70% of overall data using categorical cross entropy loss and Adam optimizer with initial lr 1e-4, beta\_1 0.9 and beta\_2 0.999. The other 30% of the overall data is used as validation dataset for model evaluation and model selection.



FigX. Network architecture used in ARC(comments, use different color for 1.0 and 0.618 too)

To select a model that fits the task, we first realized that commonly used accuracy metric cannot be used because the cost of making a mistake varies across different classes. For example, when the ground truth is decreased learning rate, an increased learning rate prediction should be penalized more than a constant learning rate prediction. Therefore, propose a reward and penalty matrix as Table X shows then compute a weighted accuracy (Wacc) based on the weights from each reward/penalty using Equation X.

|  |  |  |  |
| --- | --- | --- | --- |
| Predict \ Actual | decrease | stay | increase |
| decrease | **High reward (3)** | Low penalty (1) | **High penalty (3)** |
| stay | Low penalty (1) | Low reward (1) | Low penalty |
| increase | **High penalty (3)** | Low penalty (1) | **High reward (3)** |

Table 1 reward and penalty matrix with corresponding weights



After the best model is selected using the custom metric, it is then ready to be deployed for new tasks. The only parameter that needs to be pre-defined is n - frequency of learning rate change in terms of epochs. if a closely controlled learning rate is desired, then n can be as small as one. In general, a good rule of thumb is to set n to be one tenth of total number of epochs.

Since the change of learning rate is only required every n epoch, and model’s footprint is much smaller than modern deep networks, the impact on training speed and memory consumption is negligible.

**2.5 Relation between ARC and Adaptive learning rate optimizers**

Since our LSTM model can change learning rate an adaptive way, it’s worth mentioning the connections and differences between the lstm model and the adaptive optimizers like Adam, RMSprop [].

The update term is a result of multiplication by two terms: learning rate and corrected gradient. Adaptive optimizers focus on the corrected gradient term by monitoring the property of gradients over time. When multiple updates have inconsistent directions, the numerical scale of gradient is automatically corrected to be smaller. On the other hand, if there are multiple updates pointing towards same direction, the scale of corrected gradient will increase. The change of corrected gradient is directly reflected on update term and is sometimes interpreted as “learning rate change” even though leaning rate term itself has not changed.

Although adaptive optimizers can scale the update term in an adaptive way, modern deep learning applications rely more and more on external learning rate scheduler because adaptive optimizers have its own limitations – it cannot capture important indicators outside of gradient. For example, validation loss is very important indicator of learning rate change, decreasing learning rate when validation loss hits a plateau can often help land a better optimum. Training loss also contains useful information on whether the current learning rate is diverging the loss or not.

Therefore, ARC and adaptive optimizers are complementary of each other. The adaptive optimizers can monitor gradient terms and prevent oscillation by correcting the direction and scale of gradients. In the meantime, ARC monitors the training loss, validation loss and previous learning rate history for additional change of scale in update term. As a result, the model can optimize more efficiently with this additional layer of control as we will show in experiment section next.

**3.Experiments**

In this section, we test how autonomous learning rate scheduler can guide a training task with unseen datasets and architectures. Specifically, we deploy our ARC system on two computer vision tasks one NLP task.

**3.1 Experiment setup**

For each task, we are comparing ARC against 3 baselines: 1) BaseLR, which is constant learning rate throughout the training. 2) Cyclic cosine annealing[]. 3) Exponential decay [].

In order to gain a holistic view of the effectiveness of different schedulers, we use 3 different initial learning rates for each task. At last, to gain statistical insights on the results, we repeat each experiment 5 times and calculate its statistical properties.

For every experiment, two metrics are used to measure the success of any learning rate scheduler:

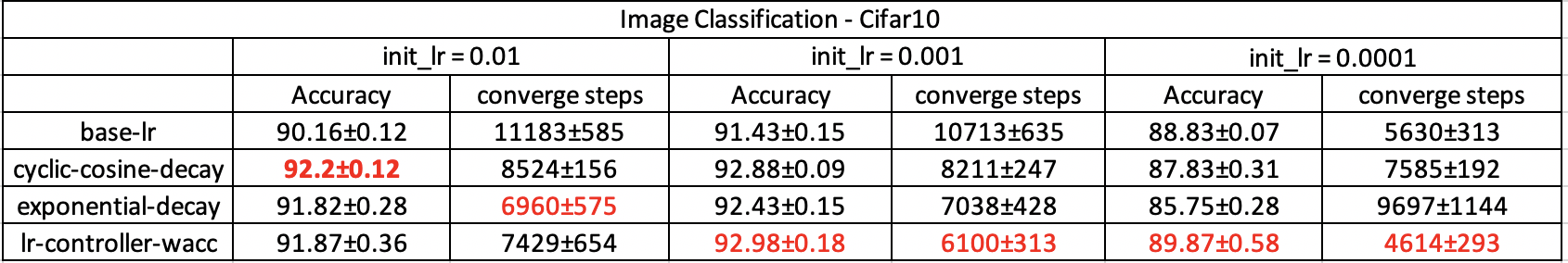
(a) Best validation performance: this is measured by task-specific metric, for example accuracy in image classification task.

(b) Convergence speed: this is measured by the number of training steps needed to reach a pre-defined validation result target. The target is defined as follows: for a specific task and initial learning rate, we have 4 learning rate schedulers each running 5 times. The target is the minimal performance among results in (a) for all 20 runs.

**3.2 Image Classification on Cifar10**

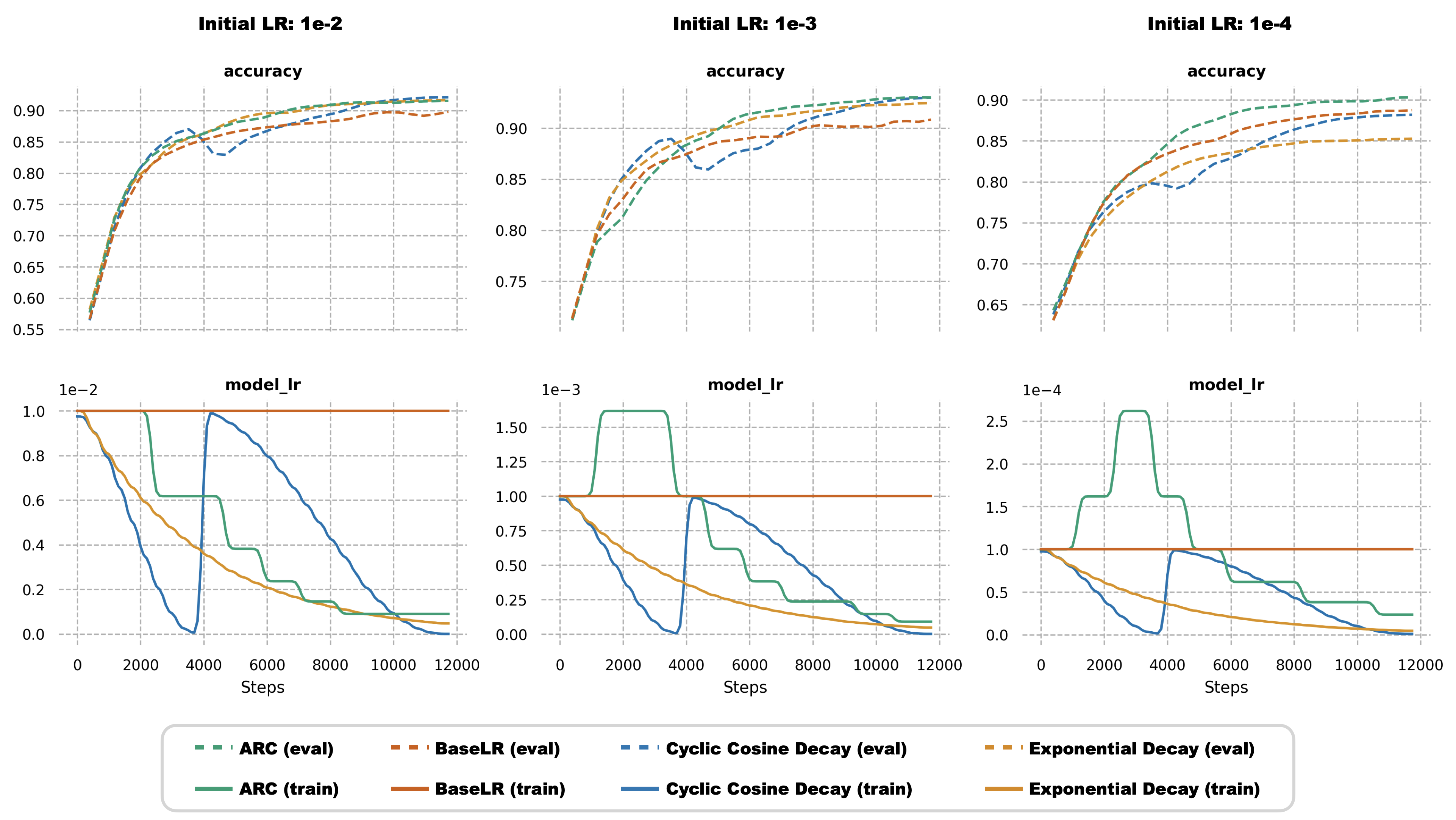
We use the same architecture and preprocessing as the one listed proposed here[ref] for a fast CIFAR 10 training. The official training uses 24 epochs but we trained for 30 epochs to ensure the model being fully trained. We use Adam optimizer[] and batch size 128. Three different initial learning rates are used: [1e-2, 1e-3, 1e-4]. For cyclic-cosine annealing parameters, we use the same setting used by the ENAS[reference] for CIFAR10 task. For exponential decay, we use a gamma of 0.9. For automatic learning rate scheduler, we change the learning rate every 3 epochs.

The Results for all experiment runs are summarized in Table X, their corresponding learning rate and accuracy over time is shown in Figure X.



When the initial learning rate is big enough (1e-2 and 1e-3), all decaying learning rate schedulers outperform constant learning rate. From Figure X(a) and (b), we can see the ARC made a series of decreasing learning rate decisions based on its inputs. As a result, among all learning rate schedulers, ARC ranked #2 with bigger initial learning rate (1e-2) and ranked #1 with medium initial learning rate (1e-3).

When learning rate is small (1e-4), however, we are starting to see drawbacks of static decaying learning rate schedulers – decaying an already small learning rate makes it difficult to converge. This is why the two decaying learning rate schedulers are worse than constant learning rate. On the other hand, as shown in Figure X (c), our lr scheduler is able to sense the learning rate being too small and increase the learning rate in the beginning phase which led to best convergence speed and ultimately best accuracy.



(need to add (a), (b),(c ))

**3.3 Object Detection on MSCOCO**

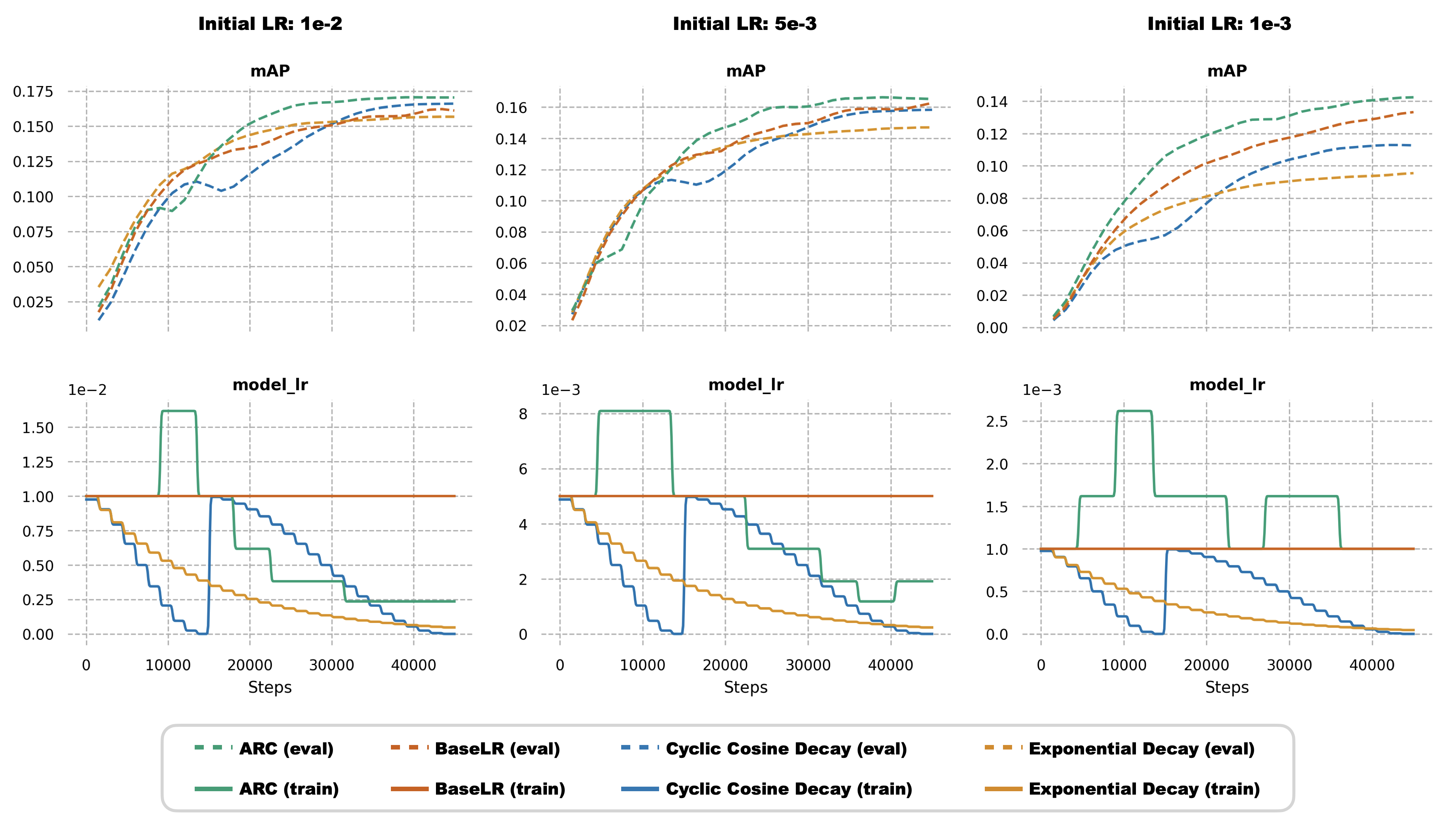
Object detection task is the most time-consuming task among the three because of the model complexity and dataset size. To ensure all 120 runs can finish within reasonable amount of time, we trained on MSCOCO dataset with size 256 as longest side. For architecture, we use RetinaNet[ref] with pretrained ResNet50[ref] as backbone. We used 32 as batch size and trained in total of 45000 steps with validation happening every 1500 steps. The optimizer used is momentum optimizer with 0.9 as momentum. The rest of the parameters are kept the same as the official implementation. The initial learning rates are [0.01, 0.005, 0.001]. The configurations of learning rate schedulers are the same as CIFAR10. mean Average Precision(mAP) is used to measure model performance on validation set.

The object detection results are summarized in Table X, the change of mAP and learning rate over time is shown in Figure X. We found that the acceptable initial learning rate range is narrower than that of CiFAR10 and any learning rate outside of the range either has a risk of divergence or converges too slow.

Interestingly, in this task, the largest learning rate (1e-2) we used does not seem to be large enough to help exponential decay outperform constant learning rate. As mentioned earlier, any greater initial learning rate may diverge the training loss. Therefore, it exposes a critical limitation of exponential learning rate decay – the rate of decay needs to be carefully tuned otherwise the learning rate can become both too high in the early training steps and too small in later training steps. On the other hand, both cyclic cosine decay and lr controller outperform constant learning rate, however, lr-controller is able to reach pre-defined performance target with much less training steps than cyclic cosine decay and achieve the best mAP in the end too.

For the other two smaller learning rates (5e-3 and 1e-3), similar to CIFAR 10, both learning rate schedulers are worse than constant learning rate because the static learning rate schedulers are unaware of a small initial learning rate. In contrast, lr controller can detect the issue and adjust the learning rate accordingly. In the end, lr controller produces the best result and requires the least amount of training steps to reach target performance.



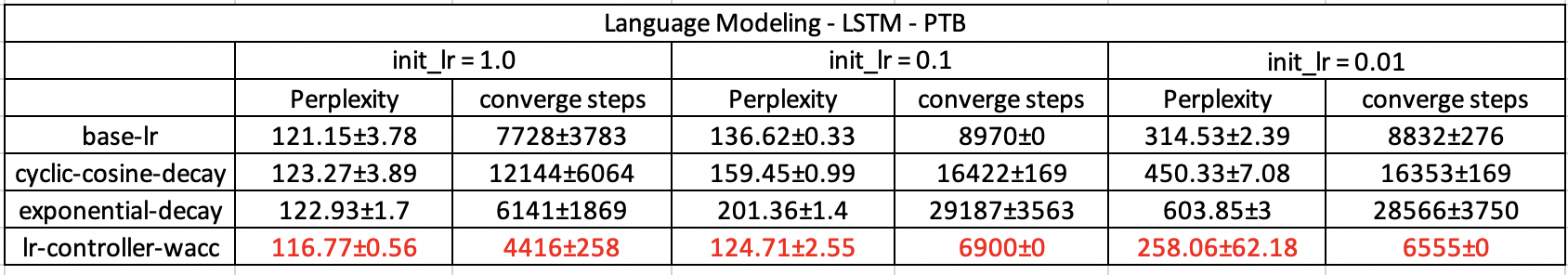


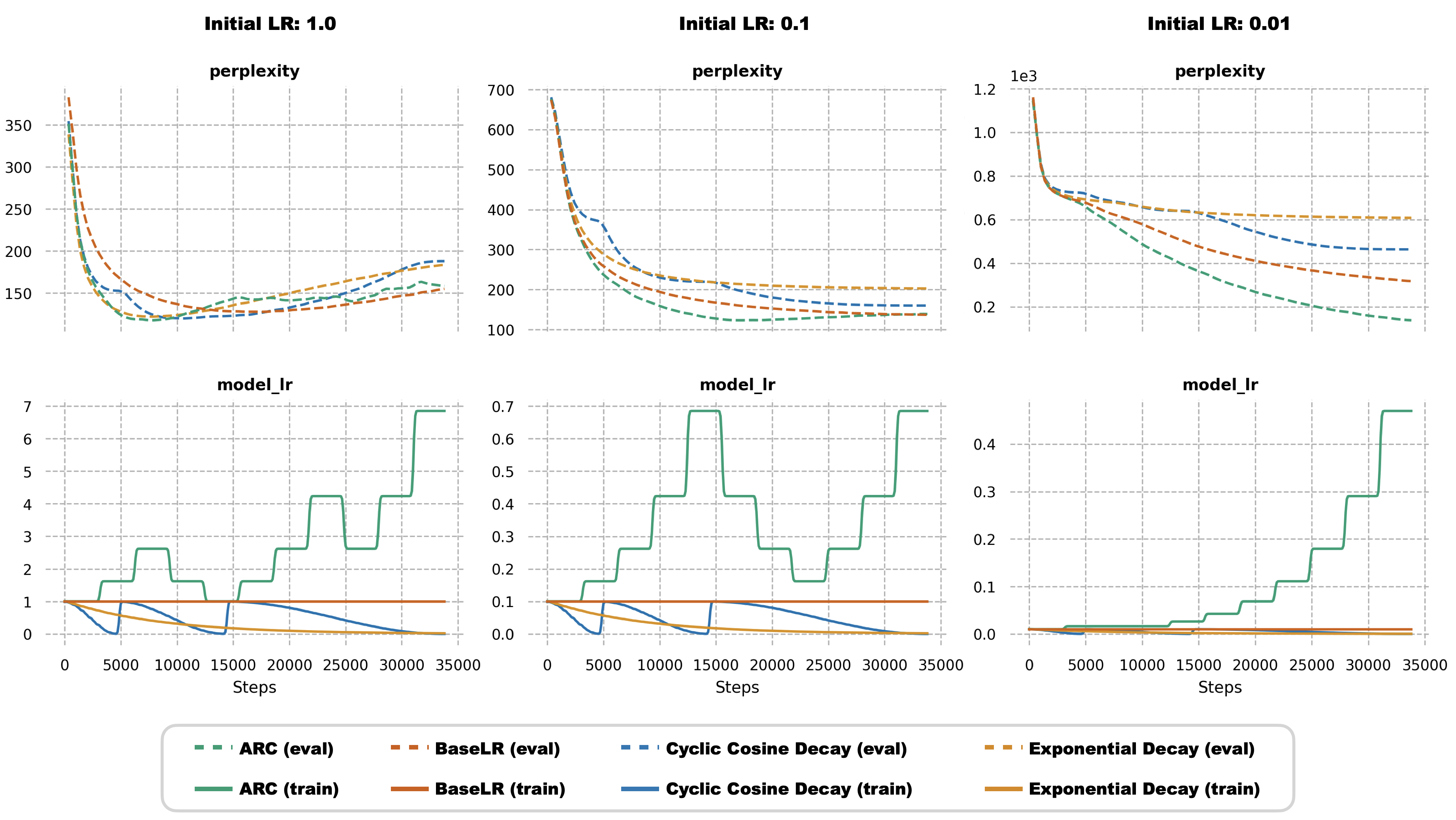
**3.4. Language Modeling on PTB**

Next, we want to extend beyond computer vision problems and see if ARC can work in language tasks too. The task is language modeling on PTB dataset with 10000 as vocabulary size. For architecture, we use 600 LSTM units with 300 embedding dimensions, a drop out with probability 0.5 is applied before the final prediction. The batch size is 128 and sequence length is 20. The initial learning rates are [1.0, 0.1, 0.01], total number of epochs we trained is 98 and optimizer is Stochastic Gradient Descent (SGD). For cyclic cosine decay, we set T\_0 as 14 epochs and T\_multi=2 so we can fit 3 learning rate cycles in the training. For exponential decay parameter, we increased gamma to 0.96 to account for larger number of epochs. For lr controller, the learning rate is changed every 9 epochs.

The language modeling results are summarized in Table X and Figure X. Perplexity is the metric we use to measure the performance of language model, and the lower the perplexity, the better model performance is. We selected our initial learning rate by factor of 10 and we made sure 1.0 is the largest learning rate in factor of 10 that we can use without loss divergence.

Surprisingly, even the biggest initial learning rate is not able to help both decaying learning rate schedulers beat the constant learning rate this time. This situation is similar to what we see in object detection with exponential decay on big learning rate. In comparison, ARC shows how learning rate changes can improve the training results significantly, especially in Figure X (c ) when it realizes the initial learning rate is too small and decided to keep increasing learning rate which ended up with the best result and the best convergence speed.





**3.6 Limitations and failure modes**

So far, we have seen many success stories of ARC, in this section we will present its limitations, as well as its failure modes from the “success story” we presented earlier.

To begin with, one limitation of ARC is that it requires static loss function, in other words, the optimization objective should not change. In contrast, many generative adversarial networks (GAN)-related training involves the training of a “moving target” because the discriminator is involved in the loss of generator but the weight of the discriminator keeps changing. Therefore, the behavior of ARC on GAN is unknown. Fortunately, the objective of most real-world problem-solving tasks can be clearly defined mathematically such that ARC can be applied.

Another limitation of current design of the ARC workflow is that it still requires an input parameter – the learning rate change frequency. A true autonomous learning rate system should be able to predict the time for next learning change. This is something that can be improved in future work of the ARC system.

For failure modes, just like any other deep learning model, the model used in ARC is not perfect even with multiple enhancements we presented in section 2. Therefore, mistakes can be made during the training and we are going to show two of these scenarios from the experiments we presented earlier.

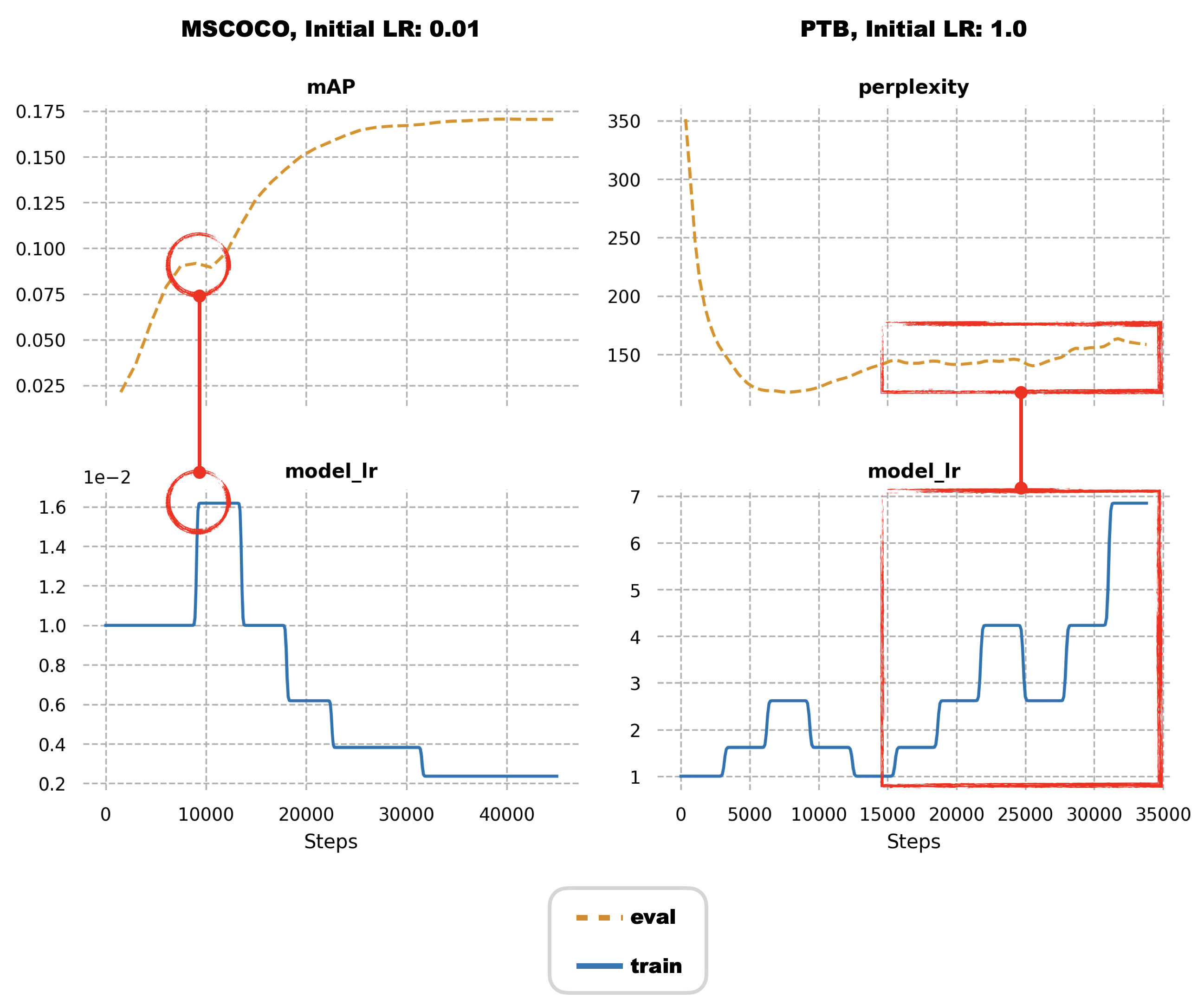


Figure X (a) shows the object detection training progress of ARC with 1e-2 as initial learning rate, as we can see, the controller makes an incorrect decision of increasing learning rate when learning rate is already big enough. As a result, we observed a big drop in mAP during validation. This is likely due to the fact that the input information is limited during early phase of the training. Luckily, the drop in validation metric is captured automatically by ARC so it realized the mistake and made series of learning rate decreases to compensate.

Figure X (b) illustrates ARC’s language modeling training with 1.0 as initial learning rate. From the validation metric curve, we saw that the ARC functions as expected in the early phase of training. For example, several learning rate increase moves are made due to great improvement observed in validation perplexity, then learning rates are decreased as perplexity hits plateau. However, after realizing that a decreased learning rate was not helpful in improving the validation results, ARC decides to keep increasing learning rate, which ended up pushing the model away from a good optimum. THIS MIGHT NOT BE A FAILURE MODE. It might be actively preventing overfitting.

**Conclusion**

In this work, we proposed an autonomous system named ARC that can automatically change learning rate in deep learning training for a better results and faster convergence. ARC addresses several challenges in learning rate scheduling and can be complementary of modern adaptive optimizers. In experiment, ARC is compared against other popular learning rate schedulers with different learning settings across various deep learning tasks, as a result, ARC demonstrated better performance as well as faster convergence speed in every task. We also illustrated limitations of ARC and potential improvement for future. The true intent of automation system in general is not about being superior than every hand-crafted system ever existed, instead, it only needs to achieve similar performance to justify its efficiency. Given that, the fact that ARC can outperform other popular human-designed learning rate schedulers has already proven itself useful.

[1] Sepp Hochreiter; Jürgen Schmidhuber (1997), "LSTM can Solve Hard Long Time Lag Problems", Advances in Neural Information Processing Systems 9, Wikidata Q77698282

https://github.com/davidcpage/cifar10-fast