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

Anonymous CVPR 2021 submission

Paper ID 5399

Abstract

There is a parameter ubiquitous throughout the deep learning world: learning rate. There is likewise a ubiquitous question: what should that learning rate be? The true answer to this question is often tedious and time consuming to obtain, and a great deal of arcane knowledge has accumulated in recent years over how to pick and modify learning rates to achieve optimal training performance. Moreover, the long hours spent carefully crafting the perfect learning rate can come to nothing the moment your network architecture, optimizer, dataset, or initial conditions change ever so slightly. But it need not be this way. We propose a new answer to the great learning rate question: the Autonomous Learning Rate Controller. Source code will be made publicly available.

1. Introduction

Learning Rate (LR) 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, LR needs to be carefully chosen. Overly large LRs will cause divergence whereas small LRs train slowly and may get trapped in a bad local minima. As training schemes have evolved over time they have begun to move away from a single static LR and into scheduled LRs, as can be seen in a variety of state-of-the-art AI applications[31][2][37][24]. LR scheduling provides finer control of LRs by allowing different LRs to be used throughout the training. However, the extra flexibility comes with a cost: these schedules bring more parameters to tune. Given this tradeoff, there are broadly two ways of approaching LR scheduling within the AI community.

Experts with sufficient compute resources tend to handcraft their own LR schedules, because a well-customized LR schedule can often lead to improvements over current state-of-the-art results. For example, entries in the Dawnbench[5] are known for using carefully tuned LR schedules to achieve world-record convergence speeds. However, such LR schedules come with significant drawbacks. First, these LR schedules are often specifically tailored to an exact problem configuration (architecture, dataset, optimizer, etc.) such that they do not generalize to other tasks. Moreover, creating these schedules tends to require a good deal of intuition, heuristics, and manual observation of training trends. As a result, building a well-customized LR schedule often requires great expertise and significant computing resources.

In contrast, others favor existing task-independent LR schedules since they often provide decent performance gains with less tuning efforts. Some popular choices are cyclic cosine decay[25], exponential decay, and warmup[10]. While these LR schedules can be used across different tasks, they are not specially optimized for any of them. As a result, these schedules do not guarantee performance improvements over a consant LR. On top of that, many of these schedules still require significant tuning to work well. For example, in cyclic cosine decay, parameters such as l_{max}, l_{min}, T_0, and T_{multi} must all be tuned in order to function properly.

Recent advancements in AutoML on architecture search[39][30][22] and update rule search[1] have proved that it is possible to create automated systems that perform equal or better than human experts in designing deep learning algorithms. These successes have inspired us to tackle the LR scheduling problem. We aim to create a system that learns how to change LR effectively.

To that end we introduce ARC: an Autonomous LR Controller. It takes training signals as input and is able to intelligently adjust LRs in a real-time generalizable fashion. ARC overcomes the challenges faced by prior LR schedulers by encoding experiences over a variety of different training tasks, and by dynamically responding to each new training situation so that no manual parameter tuning is required.

ARC is also fully complementary to modern adaptive optimizers such as Adagrad[8] and Adam[18]. Adaptive optimizers compute updates using a combination of LR and 'adaptive' gradients. When gradients have inconsistent directions across steps, the scale of the adaptive gradient is reduced. Conversely, multiple updates in the same direction result in gradient upscaling. This is sometimes referred to as adaptive LR even though the LR term has not actually been modified. Our method is gradient agnostic and instead leverages information from various training signals to directly modify the optimizer LR. This allows it to detect patterns which are invisible to adaptive optimizers. Thus the two can be used in tandem for even better results.

The key contributions of this work are:

1. An overall methodology for developing autonomous LR systems, including problem framing and dataset construction.
2. A comparison of ARC with popular LR schedules across multiple computer vision and language tasks.
3. An analysis of failure modes and unexpected behaviors from ARC, informing future directions for research.

2. Challenges and Constraints

Before delving into our methodology, we will first highlight some of the key challenges in developing an autonomous LR controller. These constraints inform many of our subsequent design decisions.

a) **Subjectivity.** Determining the superiority of one model over another (each trained with a different LR) is fraught with subjectivity. There are many different ways to measure model performance (training loss, validation loss, accuracy, etc.) and they may often be in conflict with one another.

b) **Cumulativeness.** Associating the current model performance with an LR decision at any particular step is challenging, since the current performance is the result of the cumulative effect of all previous LRs used during training.

c) **Randomness.** Randomness during training makes it difficult to compare two alternative LRs. Some common sources of randomness are dataset shuffling, data augmentation, and random network layers such as dropout. Any performance differences due to the choice of LR need to be large enough to overshadow these random effects.

d) **Scale.** Different deep learning tasks use different metrics to monitor training. The most task-independent of these are training loss, validation loss, and LR. Unfortunately, the magnitude of these values can still vary greatly between tasks. For example, categorical cross entropy for 1000-class classification is usually between 0 and 10, but a pixel-level cross entropy for segmentation can easily reach a scale of several thousand. Moreover, a reasonable LR for a given task can vary greatly, from 1e-6 up to 10 or more.

e) **Footprint.** The purpose of having an automated LR controller is to achieve faster convergence and better results. Any solution must therefore have a small enough footprint that using it does not adversely impact training speed and memory consumption.

3. Methods

3.1. Framing LR Control as a Learning Problem

We frame the development of ARC as a supervised learning problem: predicting the next LR given available training history. Due to challenge b), the model needs to observe the consequence of a specific LR for long enough to form a clear association between LR and performance. We therefore only modify the LR on a per-epoch basis.

Per challenges a) and d), as well as the desire to create a generalizable system, we cannot use any task-specific metrics. We also cannot rely on model parameters or gradient inspection since ARC would then become architecture dependent and would likely also fail constraint e). We therefore leverage only the historical training loss, validation loss, and LR as input features.

Due to challenges c) and d), rather than generating a continuous prediction of what new LR values should be, we instead pose this as a 3-class classification problem. Given the input features, should the LR: increase ($LR*1.618$), remain the same ($LR*1.0$), or decrease ($LR*0.618$)? There is no theoretical basis for our choice of multiplicative factors here, but one increase followed by one decrease will leave you roughly where you started.

3.2. Generating the Dataset

Having framed the problem, we now need to generate a dataset on which we can train ARC. To do this we need data from real deep learning training tasks. For each task we used the following procedure to generate data:

1. Train n epochs with $LR = r$, then save the current state as checkpoint C
2. Reload C, train for n epochs with $LR = 1.618 * r$, then compute validation loss (l_+)
3. Reload C, train for n epochs with $LR = 1.0 * r$, then compute validation loss (l_1)
4. Reload C, train for n epochs with $LR = 0.618 * r$, then compute validation loss (l_-)
5. Note the LR which resulted in $min\{l_+, l_1, l_-\}$
6. Reload C, eliminate $max\{l_+, l_1, l_-\}$ and its corresponding LR, replace r with one of the two remaining LRs at random, and return to step 1

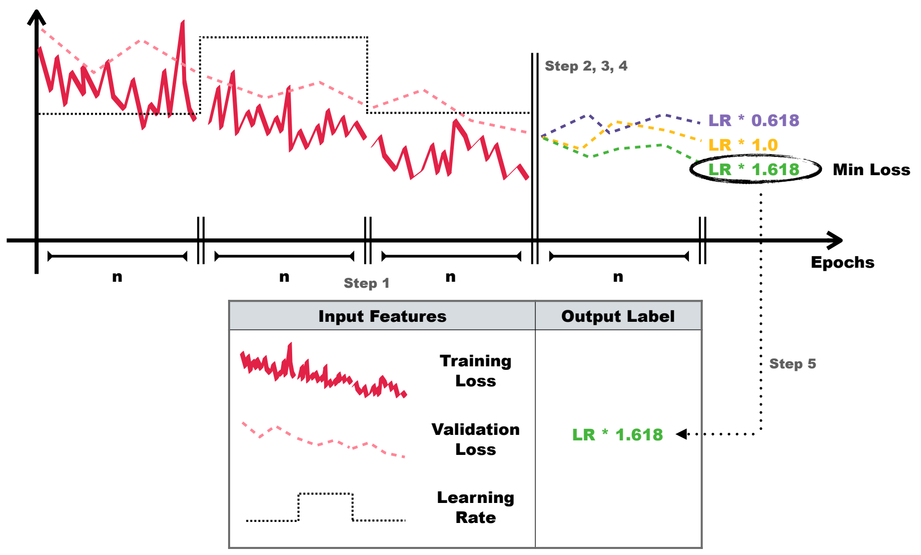

Figure 1. Generating and Labeling a Data Point

By executing steps 1 - 5 we can create one input/ground truth pair. The input features are the training loss, validation loss, and LR during step 1, concatenated with those same features from the previous $2n$ epochs of training. The label is the LR noted in step 5. This process is depicted in Figure 1.

Steps 1 - 6 continue until training finishes. Assuming the total number of training epochs is N, then we get N/n data points from each training procedure. The random selection process in step 6 is used to explore a larger search space without causing the loss to diverge.

To address challenge d), for each instance of input data we apply z-score normalization to the training and validation losses. LR is normalized by dividing by its first value. All three features are then resized to length 300 using inter-nearest interpolation.

To help ensure generalization, we gathered 12 different computer vision and language tasks - each having a different configuration (dataset, architecture, initial LR, etc.) as shown in Table 1. Each of the 12 tasks were trained an average of 42 times. Each training randomly selected an optimizer from $\{$Adam, SGD, RMSprop[35]$\}$, an initial LR $r \in [\text{Min Init LR}, \text{Max Init LR}]$, a value of $n \in [1, 10]$, and then trained for a total of $10n$ epochs. Thus approximately 5050 sample points were collected in total.

3.3. Correcting Ground Truth

Suppose that during step 5 of the data generation process you find that l_+, l_1, and l_- are 0.113, 0.112, and 0.111 respectively. Due to challenge c) it may not be appropriate to confidently claim that decreasing LR is the best course of action.

Luckily, there is one more datapoint we can use to reduce uncertainty. Suppose that during step 6 we choose to decrease the LR. Then the subsequent step 1 is repeating exactly the prior step 4. Let l_-^* be the validation loss at the end of step 1. If the relative order of l_+, l_1, and l_- is the same as the relative order of l_+, l_1, and l_-^*, then we consider our ground truth labeling to be correct (for example, if $l_-^* = 0.109$). On the other hand, if the relative ordering is different (for example, if $l_-^* = 0.115$), then random noise is playing a greater role than the LR in determining performance. In that case we take a conservative approach and modify the ground truth label to be 'constant LR'.

3.4. Building the Model

The ARC model architecture is shown in Figure 2. It consists of three components: a feature extractor, an LSTM[12], and a dense classifier. The feature extractor consists of two 1D convolution layers, the LSTM of two stacked memory sequences, and the classifier of two densely connected layers. Considering constraint e), we chose layer parameters such that the total number of trainable model parameters is less than 80k. Compared to the millions of parameters which are common in current state-of-the-art models, this architecture should add relatively minimal overhead.

The model was trained with the corrected dataset from Section 3.3, split 70/30 between training and validation data. We used categorical cross entropy for the loss, and leveraged an Adam optimizer with the following parameters: $LR = 1e-4$, $beta_1 = 0.9$, and $beta_2 = 0.999$. Training proceeded with a batch size of 128 for 300 epochs, with the best model being saved along the way. We define the

Task	Task Description	Dataset	Architecture	Min Init LR	Max Init LR
1	Image Classification	SVHN Cropped[28]	VGG19[33] + BatchNorm[14]	1e-3	1e-5
2	Image Classification	SVHN Cropped	VGG16[33] + ECC[36]	1e-4	1e-5
3	Adversarial Training[9]	SVHN Cropped	VGG19	1e-3	1e-5
4	Image Classification	Food101[3]	Densenet121[13]	1e-2	1e-5
5	Image Classification	Food101	InceptionV3[34]	1e-2	1e-5
6	Multi-Task[17]	CUB200[38]	ResNet50[11] + UNet[32]	1e-4	1e-5
7	Text Classification	IMDB[26]	LSTM	1e-3	1e-5
8	Named Entity Recognition	MIT Movie Corpus[23]	BERT[6]	1e-4	1e-5
9	One Shot Learning	omniglot[20]	Siamese Network[19]	1e-3	1e-5
10	Text Generation	Shakespear[16]	GRU[4]	1e-3	1e-5
11	Semantic Segmentation	montgomery[15]	UNet	1e-4	1e-5
12	Semantic Segmentation	CUB200	UNet	1e-3	1e-5

Table 1. Training Dataset Task Overview

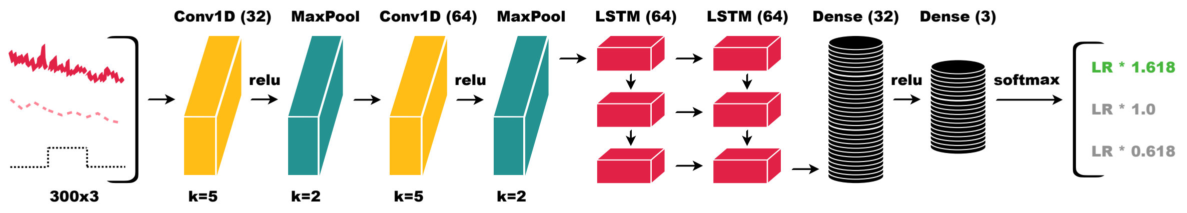

Figure 2. Network Architecture Used in ARC

Predict \Actual	Decrease	Constant	Increase
Decrease	$+3$	-1	-3
Constant	-1	$+1$	-1
Increase	-3	-1	$+3$

Table 2. Reward / Penalty Matrix

'best' model using a weighted accuracy, since predicting a constant LR is less problematic than calling for a change in the incorrect direction. The Reward/Penalty Matrix (RPM) for this metric is given in Table 2, with the corresponding metric computed from the Confusion Matrix (CM) per Equation 1.

$$w_{acc} = \frac{\sum_i CM[i,i] * |RPM[i,i]|}{\sum_{i,j} CM[i,j] * |RPM[i,j]|} \tag{1}$$

Once trained, the ARC model can be used to periodically adjust the LR for other models, as we will demonstrate in Section 4.

4. Experiments

In this section, we test how well ARC can guide training tasks on previously unseen datasets and architectures. Specifically, we deploy ARC on two computer vision tasks and one NLP task. For each task, we compare ARC against 3 standard LR schedules: Baseline LR (BLR) - in which LR is held constant, Cyclic Cosine Decay (CCD), and Exponential Decay (ED).

In order to gain a holistic view of the effectiveness of different schedulers, we use 3 different initial LRs for each task. Each training configuration is run 5 times in order to compute standard deviations. Scheduler performance is measured in two ways: best validation metric performance (e.g. highest accuracy), and fewest steps till convergence. We define convergence by running each of the 4 schedulers 5 times, finding the best metric scores for each of those 20 runs, and then picking the worst of those scores to be our convergence threshold. Schedulers which can reach that threshold the fastest are preferable. All experiments were implemented in the deep learning framework FastEstimator[7].

4.1. Image Classification on CIFAR10

For our first experiment we trained a model to perform CIFAR10 image classification. We used the same architecture and preprocessing as proposed in [29]. We trained for 30 epochs (rather than 24 in the original implementation) using an Adam optimizer and a batch size of 128. Three different initial LRs were used: 1e-2, 1e-3, and 1e-4. For each initial LR, we compare the performance of BLR with the performance of ARC (invoked every 3 epochs), as well as CCD (using the settings proposed in [30] for CIFAR10), and ED (with a gamma of 0.9).

	Initial LR = 0.01		Initial LR = 0.001		Initial LR = 0.0001	
	Accuracy	Converge Step	Accuracy	Converge Step	Accuracy	Converge Step
BLR	90.16 ± 0.12	11020 ± 640	91.43 ± 0.15	10520 ± 621	88.83 ± 0.07	5340 ± 361
CCD	$\mathbf{92.20 \pm 0.12}$	8400 ± 228	92.88 ± 0.09	8040 ± 361	87.83 ± 0.31	7460 ± 196
ED	91.82 ± 0.28	$\mathbf{6800 \pm 580}$	92.43 ± 0.15	6900 ± 415	85.75 ± 0.28	9500 ± 1207
ARC	91.87 ± 0.36	7120 ± 671	$\mathbf{92.98 \pm 0.18}$	$\mathbf{5840 \pm 224}$	$\mathbf{89.87 \pm 0.58}$	$\mathbf{4440 \pm 258}$

Table 3. Results for CIFAR10 Image Classification (best values in bold)

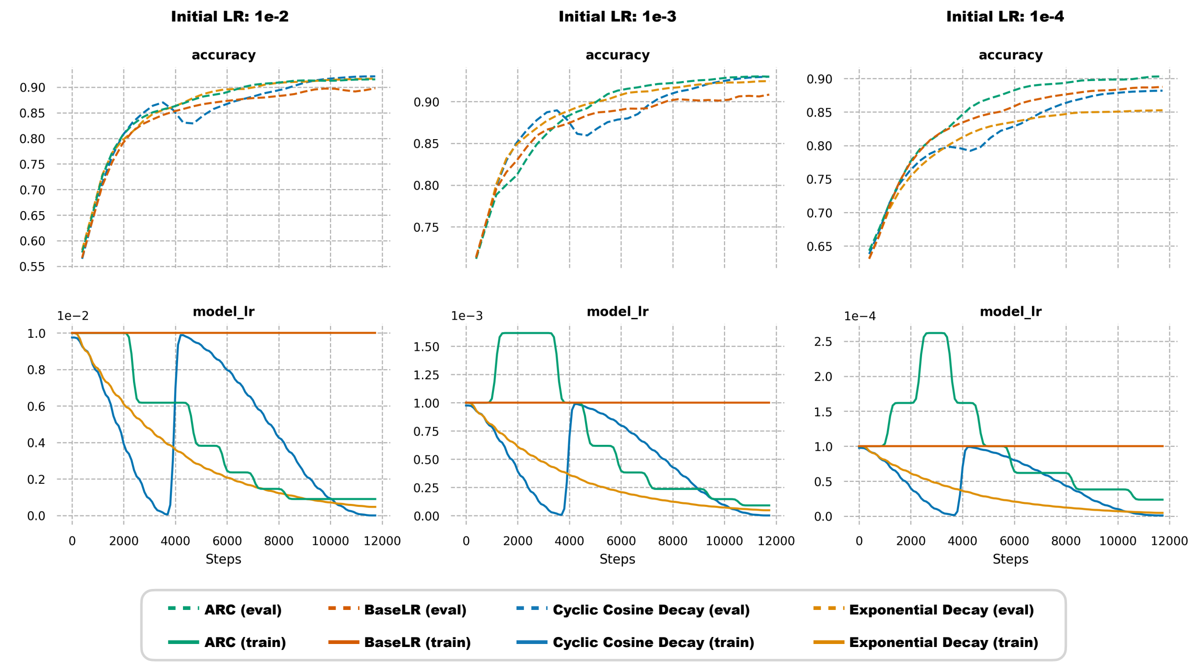

Figure 3. Performance on CIFAR10

The results for all experiment runs are summarized in Table 3. Representative graphs of LR and accuracy for each method are given in Figure 3.

When the initial LR is sufficiently large (1e-2 and 1e-3), all decaying LR schedulers outperform the baseline LR. From Figure 3 (left and center), we can see that ARC also decided to decrease the LR. Amongst all LR schedulers, ARC performed second best with the large initial LR (1e-2), and was the best performer with the medium initial LR (1e-3).

When the initial LR is small (1e-4), however, the drawback of statically decaying LR schedulers becomes evident: decaying an already small LR damages convergence. In this case, CCD and ED are beaten by the baseline LR. On the other hand, as shown in Figure 3 (right), ARC is able to sense that the LR is too small and increase it, achieving the best final accuracy and convergence speed.

4.2. Object Detection on MSCOCO

Our second and most time-consuming task is object detection using the MSCOCO dataset. We downscale the longest side of each image to 256 pixels in order to complete the trainings within a more reasonable computational budget. The RetinaNet[21] architecture was selected for this task. We used a batch size of 32 and trained for a total of 45000 steps, with validation every 1500 steps. We used a momentum optimizer with 0.9 for its momentum value, but kept all other parameters consistent with the official implementation. The configuration for our LR schedulers is the same as in Section 4.1, but with initial LRs of 0.01, 0.005, and 0.001 (we found that LRs outside of this range risked divergence or converged too slowly to be useful). For this task we use mean average precision (mAP) to benchmark model performance.

The results for all experiment runs are summarized in Table 4. Representative graphs of LR and mAP for each method are given in Figure 4.

	Initial LR = 0.01		Initial LR = 0.005		Initial LR = 0.001	
	mAP	Converge Step	mAP	Converge Step	mAP	Converge Step
BLR	0.163 ± 0.002	30380 ± 2905	0.163 ± 0.001	23140 ± 920	0.134 ± 0.001	16480 ± 519
CCD	0.168 ± 0.002	29600 ± 805	0.159 ± 0.001	28480 ± 1013	0.111 ± 0.001	24880 ± 637
ED	0.156 ± 0.001	36340 ± 5398	0.146 ± 0.001	37260 ± 4986	0.096 ± 0.001	38220 ± 4835
ARC	$\mathbf{0.171 \pm 0.002}$	$\mathbf{21440 \pm 1384}$	$\mathbf{0.165 \pm 0.005}$	$\mathbf{18820 \pm 279}$	$\mathbf{0.143 \pm 0.002}$	$\mathbf{12480 \pm 979}$

Table 4. Results for MSCOCO Object Detection (best values in bold)

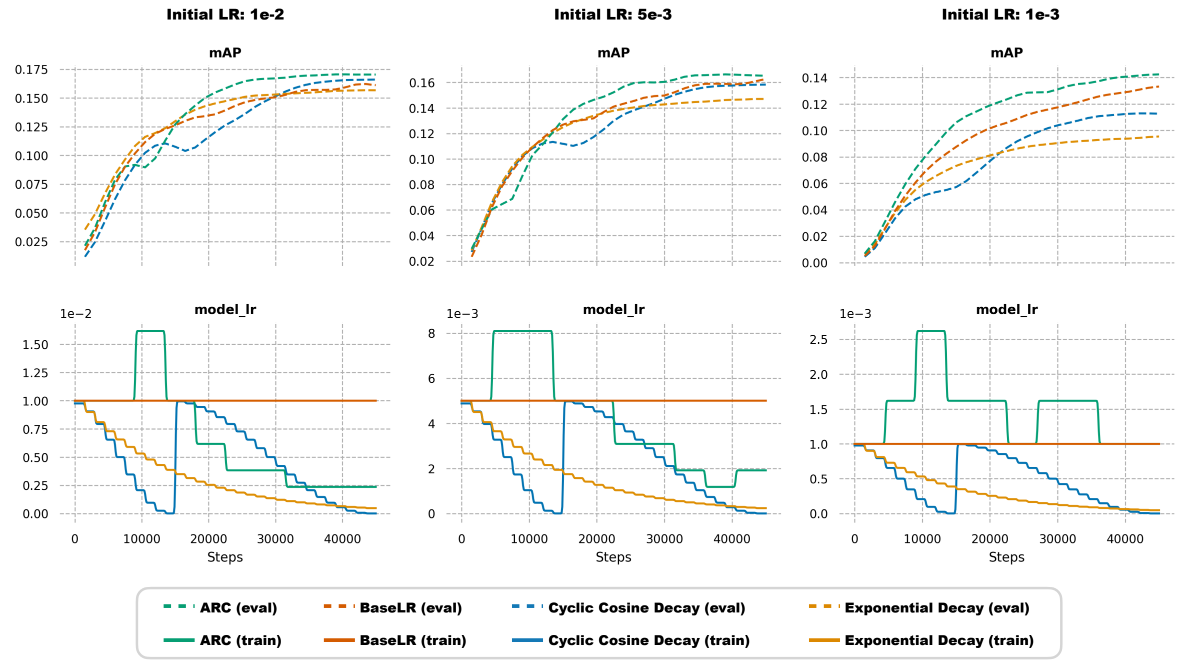

Figure 4. Performance on MSCOCO

Interestingly, the largest LR (1e-2) we used was not large enough to allow ED to outperform the baseline LR. Unfortunately, larger initial LRs were found to lead to training divergence. This exposes a critical limitation of exponential LR decay: the rate of decay needs to be carefully tuned, otherwise LR can be either too large early on or too small later in training. On the other hand, both CCD and ARC outperform the baseline LR, with ARC achieving the best mAP and dramatically better convergence speed.

For the other two smaller LRs (5e-3 and 1e-3), both ED and CCD are worse than the baseline LR because they have no mechanism to raise the LR when doing so would be useful. In contrast, ARC can notice this deficiency and increase the LR accordingly - allowing it to achieve the best mAP and convergence scores across the board.

4.3. Language Modeling on PTB

For our final task we move beyond computer vision to verify whether ARC can be useful in natural language processing tasks as well. We performed language modeling using the PTB dataset[27] with a vocabulary size of 10000. Our network for this problem leveraged 600 LSTM units with 300 embedding dimensions, and a 50% dropout applied before the final prediction. Training progressed for 98 epochs, with a batch size of 128 and a sequence length of 20. A Stochastic Gradient Descent (SGD) optimizer was selected, with initial LR values of 1.0, 0.1, and 0.01 (with 1.0 being the largest power of 10 we could find which did not lead to training divergence). Our CCD scheduler used $T_0 = 14$ and $T_{multi} = 2$ such that we could fit 3 LR cycles into the training window. The ED scheduler gamma value was set to 0.96, and ARC was invoked every 9 epochs. For this task we used perplexity to measure model performance (lower is better).

The results for all experiment runs are summarized in Table 5. Representative graphs of LR and perplexity for each method are given in Figure 5.

Surprisingly, even our largest initial LR did not allow CCD nor ED to outperform the baseline LR. The situation is similar to what we observed in Section 4.2 with ED. In con-

	Initial LR = 1.0		Initial LR = 0.1		Initial LR = 0.01	
	Perplexity	Converge Step	Perplexity	Converge Step	Perplexity	Converge Step
BLR	121.2 ± 3.8	7540 ± 3770	136.6 ± 0.3	8800 ± 63	314.5 ± 2.4	8640 ± 224
CCD	123.3 ± 3.9	11960 ± 6090	159.5 ± 1.0	16260 ± 206	450.3 ± 7.1	16160 ± 120
ED	122.9 ± 1.7	5900 ± 1909	201.4 ± 1.4	29000 ± 3592	603.9 ± 3.0	28400 ± 3769
ARC	**116.8 ± 0.6**	**4140 ± 258**	**124.7 ± 2.6**	**6840 ± 80**	**258.1 ± 62.2**	**6480 ± 40**

Table 5. Results for PTB Language Modeling (best values in bold)

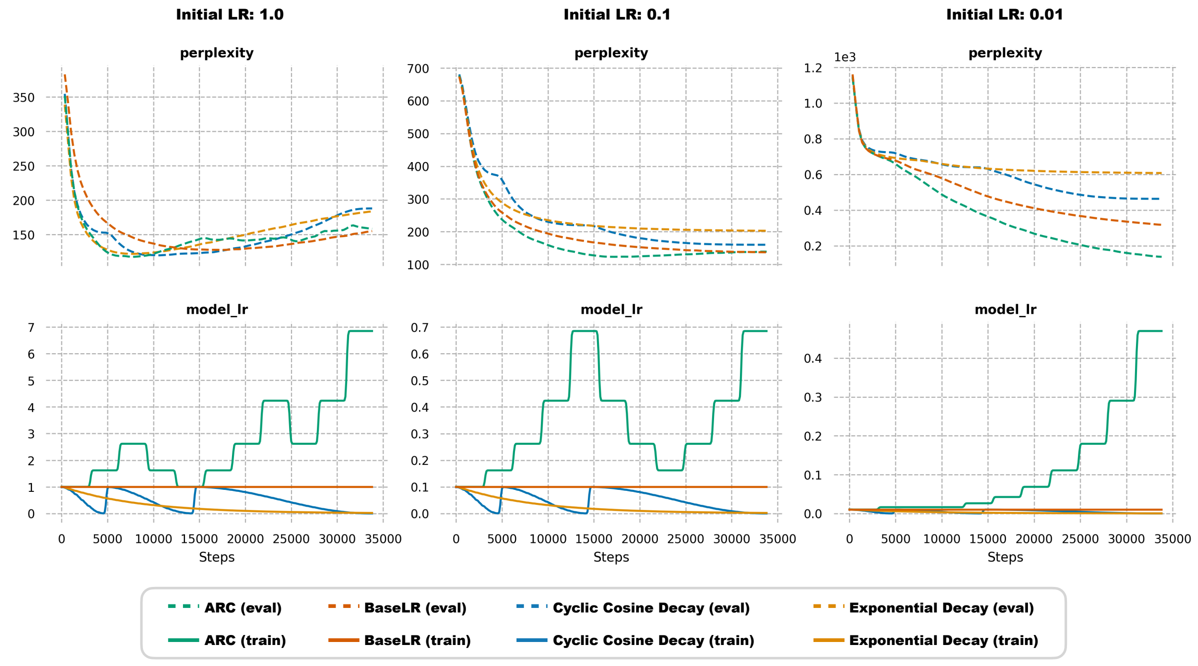

Figure 5. Performance on PTB

trast, ARC shows how LR changes can improve the training results significantly, especially in Figure 5 (right) where the system continuously increased the LR throughout training, leading to the best final performance and convergence speeds.

5. Limitations and Unexpected Behaviors

As Section 4 demonstrates, ARC can be successfully deployed over a range of tasks, architectures, optimizers, and initial LRs. It does, however, have some limitations and failure modes which bear mentioning.

One limitation of ARC is that it requires a constant optimization objective. While this is often the case for real-world problem-solving tasks, it is not true of generative adversarial networks (GANs), where the loss of the generator is based on the performance of an ever-evolving discriminator. Thus ARC, while applicable to many problems, may not be appropriate for all genres of deep learning research.

A second limitation of the current ARC implementation is that it sometimes provides unreliable decisions if queried too frequently. While we found that once per epoch typically works, running every few epochs can be more reliable. In our experiments, we chose to evenly distribute 10 invocations throughout the training. A truly autonomous system ought to determine its own update frequency. We see eliminating this parameter as an area for future research.

As for failure modes, just like any other deep learning model, ARC can also make incorrect decisions. Figure 6 (left) shows an instance of object detection training where ARC incorrectly chose to raise the LR, leading to a small decrease in mAP. Nevertheless, ARC detected the problem and was then able to course correct and drop the LR again in order to get the training back on track. This bad behavior may have been due to the fact that less historical information is available early on during training.

Figure 6 (right) shows an interesting phenomena which may or may not be a failure mode. After achieving an optimal performance around step 7500, ARC started to drop the LR as might normally be expected to improve performance.

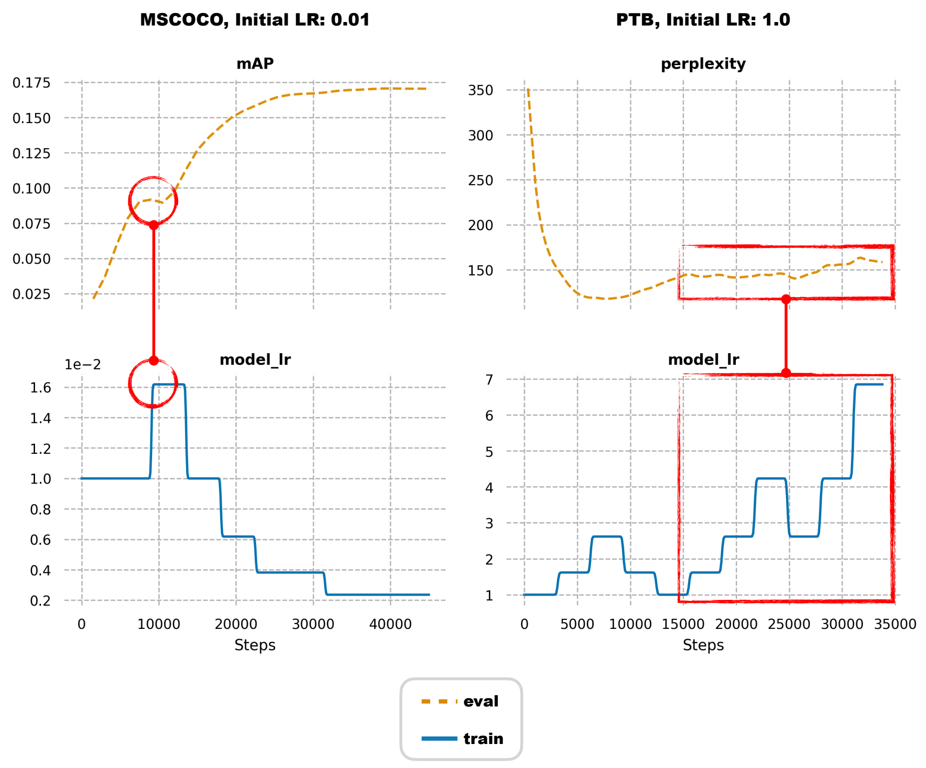

Figure 6. Unexpected Behaviors

However, after step 15000 it changed course and dramatically increased the LR, despite the fact that perplexity continued to get worse. Comparing this with what happened to the other schedulers for the same task configuration in Figure 5 (left), however, ARC might actually be attempting to prevent the model from overfitting. This is something we hope to investigate more thoroughly in the future.

6. Conclusion

In this work we proposed an autonomous learning rate controller that can guide deep learning training to reach better results in less time. ARC overcomes several challenges in LR scheduling and is complementary to modern adaptive optimizers. We experimentally demonstrated its superiority to conventional schedulers across a variety of tasks, optimizers, and network architectures, as well as identifying several areas for future improvement. Not only that, ARC achieves its objectives without tangibly increasing the training budget, in sharp contrast to other AutoML paradigms. The true test of any automation system is not whether it can outperform any possible hand-crafted custom solution, but rather whether it can provide a high quality output with great efficiency. Given that, the fact that ARC actually does outperform popular scheduling methods while requiring no effort nor extra computation budget on the part of the end user makes it a valuable addition to the AutoML domain.

References

[1] Irwan Bello, Barret Zoph, Vijay Vasudevan, and Quoc V. Le. Neural optimizer search with reinforcement learning. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pages 459–468. PMLR, 2017. 1

[2] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. Yolov4: Optimal speed and accuracy of object detection. *CoRR*, abs/2004.10934, 2020. 1

[3] Lukas Bossard, Matthieu Guillaumin, and Luc Van Gool. Food-101 – mining discriminative components with random forests. In *European Conference on Computer Vision*, 2014. 4

[4] Kyunghyun Cho, Bart van Merrienboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. In Dekai Wu, Marine Carpuat, Xavier Carreras, and Eva Maria Vecchi, editors, *Proceedings of SSST@EMNLP 2014, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation, Doha, Qatar, 25 October 2014*, pages 103–111. Association for Computational Linguistics, 2014. 4

[5] Cody A. Coleman, Deepak Narayanan, Daniel Kang, Tian Zhao, Jian Zhang, Luigi Nardi, Peter Bailis, Kunle Olukotun,

Chris Re, and Matei Zaharia. Dawnbench: An end-to-end deep learning benchmark and competition, 2017. NIPS ML Systems Workshop. 1

[6] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. In Jill Burstein, Christy Doran, and Thamar Solorio, editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 4171–4186. Association for Computational Linguistics, 2019. 4

[7] Xiaomeng Dong, Junpyo Hong, Hsi-Ming Chang, Michael Potter, Aritra Chowdhury, Purujit Bahl, Vivek Soni, Yun-Chan Tsai, Rajesh Tamada, Gaurav Kumar, Caroline Favart, V. Ratna Saripalli, and Gopal Avinash. Fastestimator: A deep learning library for fast prototyping and productization, 2019. NeurIPS Systems for ML Workshop. 4

[8] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(61):2121–2159, 2011. 1

[9] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. 4

[10] Priya Goyal, Piotr Dollár, Ross B. Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch SGD: training imagenet in 1 hour. *CoRR*, abs/1706.02677, 2017. 1

[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 770–778. IEEE Computer Society, 2016. 4

[12] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997. 3

[13] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. Densely connected convolutional networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, pages 2261–2269. IEEE Computer Society, 2017. 4

[14] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In Francis R. Bach and David M. Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, volume 37 of *JMLR Workshop and Conference Proceedings*, pages 448–456. JMLR.org, 2015. 4

[15] S. Jaeger, S. Candemir, S. Antani, Y. X. Wang, P. X. Lu, and G. Thoma. Two public chest X-ray datasets for computer-aided screening of pulmonary diseases. *Quant Imaging Med Surg*, 4(6):475–477, Dec 2014. 4

[16] Andrej Karpathy. The unreasonable effectiveness of recurrent neural networks. `http://karpathy.github.io/2015/05/21/rnn-effectiveness/`. Accessed: 2020-11-04. 4

[17] Alex Kendall, Yarin Gal, and Roberto Cipolla. Multi-task learning using uncertainty to weigh losses for scene geometry and semantics. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, pages 7482–7491. IEEE Computer Society, 2018. 4

[18] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. 1

[19] Koch, Gregory, Richard Zemel, and Ruslan Salakhutdinov. Siamese neural networks for one-shot image recognition, 2015. ICML Deep Learning Workshop. 4

[20] Brenden M. Lake, Ruslan Salakhutdinov, and Joshua B. Tenenbaum. Human-level concept learning through probabilistic program induction. *Science*, 350(6266):1332–1338, 2015. 4

[21] Tsung-Yi Lin, Priya Goyal, Ross B. Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017*, pages 2999–3007. IEEE Computer Society, 2017. 5

[22] Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: differentiable architecture search. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. 1

[23] Jingjing Liu, Scott Cyphers, Panupong Pasupat, Ian McGraw, and James R. Glass. A conversational movie search system based on conditional random fields. In *INTERSPEECH 2012, 13th Annual Conference of the International Speech Communication Association, Portland, Oregon, USA, September 9-13, 2012*, pages 2454–2457. ISCA, 2012. 4

[24] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized BERT pretraining approach. *CoRR*, abs/1907.11692, 2019. 1

[25] Ilya Loshchilov and Frank Hutter. SGDR: stochastic gradient descent with warm restarts. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. 1

[26] Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 142–150, Portland, Oregon, USA, June 2011. Association for Computational Linguistics. 4

[27] Mitchell P Marcus. Treebank-3 ldc99t42. web download. `https://catalog.ldc.upenn.edu/LDC99T42`. Accessed: 2020-11-04. 6

[28] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y. Ng. Reading digits in natural images with unsupervised feature learning, 2011. NIPS Deep Learning and Unsupervised Feature Learning Workshop. 4

[29] David C Page. Cifar10-fast dawn benchmark implementation. `https://github.com/davidcpage/cifar10-fast`. Accessed: 2020-11-04. 4

[30] Hieu Pham, Melody Y. Guan, Barret Zoph, Quoc V. Le, and Jeff Dean. Efficient neural architecture search via parameter sharing. In Jennifer G. Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 4092–4101. PMLR, 2018. 1, 4

[31] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67, 2020. 1

[32] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In Nassir Navab, Joachim Hornegger, William M. Wells III, and Alejandro F. Frangi, editors, *Medical Image Computing and Computer-Assisted Intervention - MICCAI 2015 - 18th International Conference Munich, Germany, October 5 - 9, 2015, Proceedings, Part III*, volume 9351 of *Lecture Notes in Computer Science*, pages 234–241. Springer, 2015. 4

[33] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. 4

[34] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 2818–2826. IEEE Computer Society, 2016. 4

[35] T. Tieleman and G. Hinton. Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning, 2012. 3

[36] Gunjan Verma and Ananthram Swami. Error correcting output codes improve probability estimation and adversarial robustness of deep neural networks. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, 8-14 December 2019, Vancouver, BC, Canada*, pages 8643–8653, 2019. 4

[37] Chien-Yao Wang, Hong-Yuan Mark Liao, Yueh-Hua Wu, Ping-Yang Chen, Jun-Wei Hsieh, and I-Hau Yeh. Cspnet: A new backbone that can enhance learning capability of cnn. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pages 390–391, 2020. 1

[38] P. Welinder, S. Branson, T. Mita, C. Wah, F. Schroff, S. Belongie, and P. Perona. Caltech-UCSD Birds 200. Technical Report CNS-TR-2010-001, California Institute of Technology, 2010. 4

[39] Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. 1