UNIVERSITY OF OKLAHOMA

GRADUATE COLLEGE

AUTOMATIC MACHINE LEARNING IN OPTIMIZATION AND DATA AUGMENTATION

A DISSERTATION

SUBMITTED TO THE GRADUATE FACULTY

in partial fulfillment of the requirements for the

Degree of

DOCTOR OF PHILOSOPHY

By

XIAOMENG DONG
Norman, Oklahoma
2022

AUTOMATIC MACHINE LEARNING IN OPTIMIZATION AND DATA AUGMENTATION

A DISSERTATION APPROVED FOR THE
GALLOGLY COLLEGE OF ENGINEERING

BY THE COMMITTEE CONSISTING OF

Abstract

This dissertation introduces Automatic Machine Learning (AutoML) as a potential approach to overcome current deep learning challenges on efficiency and cost. It also proposes two novel AutoML workflows to areas in deep learning where AutoML is less recognized by the AI community: optimization and data augmentation.

The proposed AutoML workflow in optimization can automatically adjust the learning rate for deep learning tasks. It monitors the signals generated during optimization and dynamically changes the learning rate based on the signal observed. The workflow is successfully deployed in image classification, instance detection and language modeling tasks. The method delivers better performance and faster convergence speed than widely-used static and learning-based schedulers under various different settings.

The new AutoML workflow in data augmentation can help deep neural networks achieve better generalization performance through automated optimization of data augmentation policies. Comparing with the prior best method, the workflow halves the computation required while achieving equivalent or better results on the same benchmarks. In addition, it also removes the need of human intervention in the workflow, making the workflow truly automated for deep learning applications.

Finally, the dissertation concludes that AutoML can play a significant role on various aspects of deep learning through further efficiency improvement and cost reduction. The hope of the dissertation is to inspire more AutoML research on all areas of deep learning, so that AutoML can eventually facilitate the development of fully automated learning, an important milestone in our long pursuit of Artificial Intelligence that can lead us to a brighter future.

Acknowledgement

I would like to thank my advisor, Dr.Trafalis, for his immense support during my entire graduate student career. We met long before I even enter the field of data science, and we discussed about everything from kernel methods to deep learning. Throughout the years, I enjoy and cherish every moment of our discussion, your insights and knowledge have always been my inspiration to become better and pursue higher goals.

I would also like to thank Dr.Hougen, Dr.Grant, Dr.Zheng, and Dr.Shehab for all the feedbacks provided throughout my dissertation research. Their advice and support have helped me to a great extent towards improving this work. The knowledge I learned from their courses and project have become the foundation of this study and my future research.

I owe a deep gratitude to Dr.Shehab, Dr.Sridhar, and Nicola Manos. They have provided me continuous support on my degree and helped me walk through the journey smoothly. My work would not have been possible without their support.

Table of Contents

List of Tables

List of Figures

1 Introduction

Deep learning has gained tremendous amount of success over the last decade and is now playing a significant role in almost every aspects of our daily life (Figure 1). For example, recommendation systems of many online shopping, video, music, and TV streaming services are built by deep learning. Fraud detection service in credit card companies and scam detection in our email systems are also powered by deep learning technology. Even our cellphone nowadays has built-in deep learning algorithms that help produce better photo quality [1] (Apple Machine Learning Research, 2021).

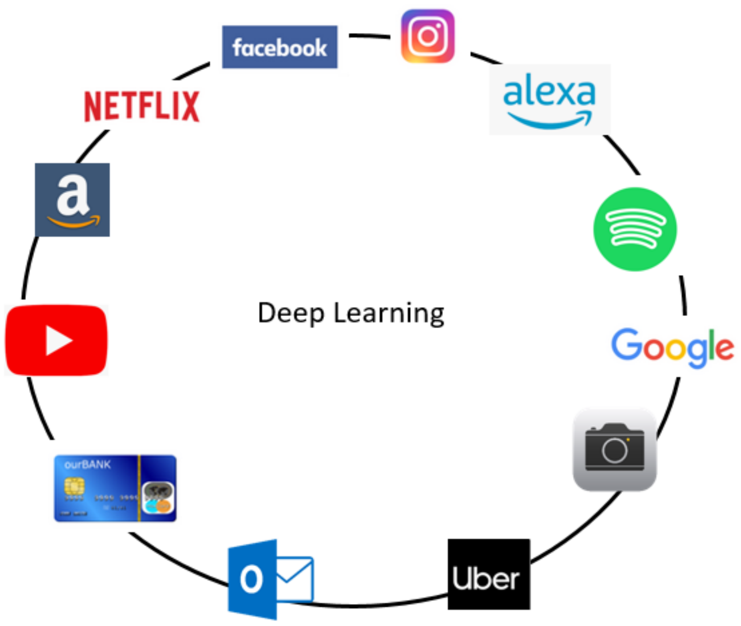

Figure 1 Deep Learning Applications in Daily Life

The great success of deep learning applications comes from its ability to excel in many fundamental tasks. In computer vision, it achieved above-human performance at object recognition, semantic segmentation, instance detection, key point detection, instance segmentation, and many other tasks [2] (Paper with Code, 2022). In Natural Language Processing (NLP), deep learning has reached near-human or above-human performances at machine translation, language modeling, semantics understanding, and more [3] (SuperGLUE Benchmark, 2022). These successes are also accompanied by a significant increase of research interest over the recent years. For example, the number of deep learning related research papers in Neural Information Processing Systems Conference (NeuIPS) has increased exponentially (Figure 2) since 2010s, with no sign of it ever slowing down in the near future.

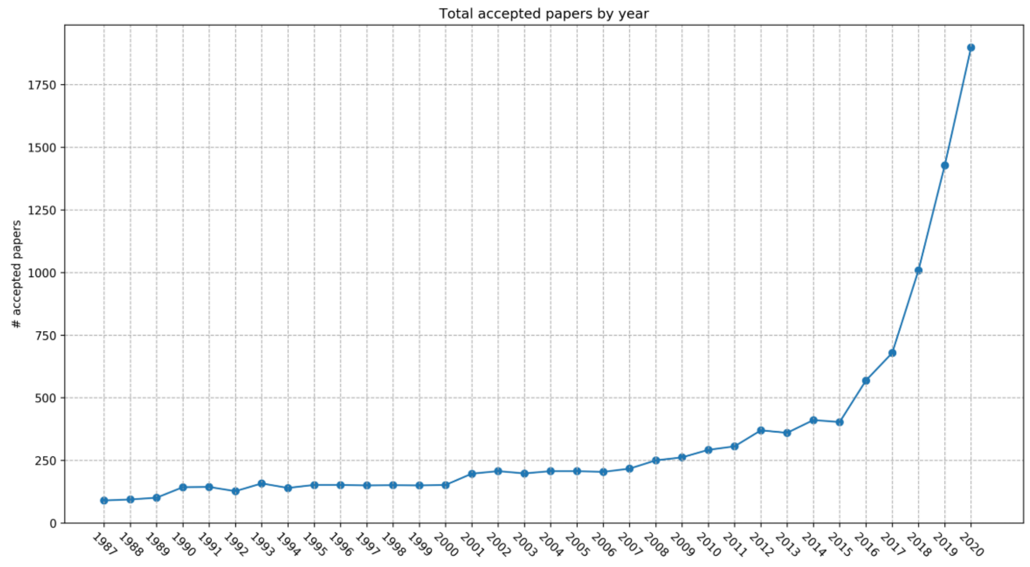

Figure 2 Number of Papers Accepted in NeurIPS (NIPS) Conference Each Year

There is no doubt that deep learning has played an important role in many aspects of our life, however, the sudden success of deep learning also comes with a dark side. The most alarming concern about deep learning is its empirical nature, such that many things people do in deep learning have little theoretical reasons other than “it just works”, making the knowledge discovery a pure trial and error process. This is why deep learning is also sometimes described by community as “stirring a pile of sand until it starts to look like a castle” (Figure 3). As a result, the empirical nature poses three major issues that hinder the advancement of deep learning.

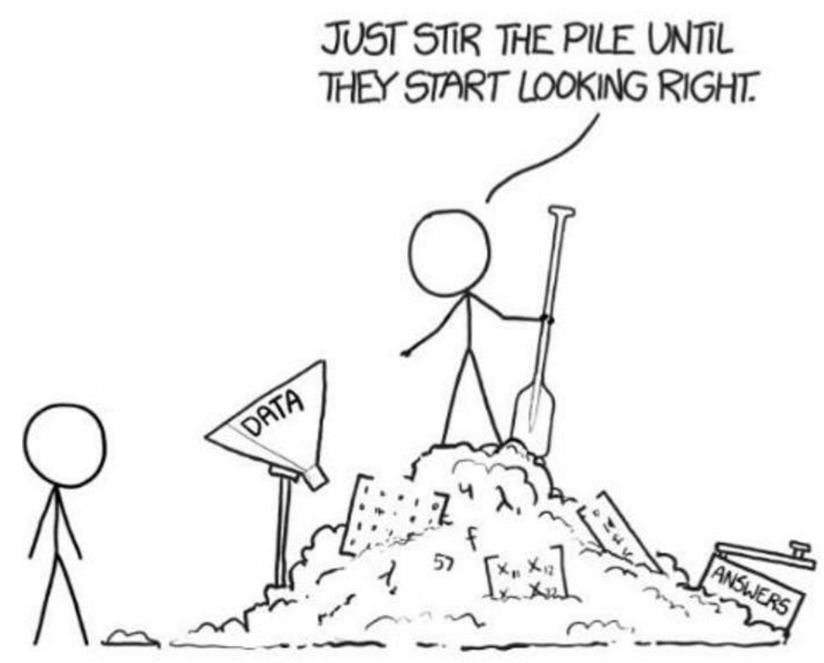

Figure 3 Comical Description of Deep Learning (From XKCD)

The first issue brought by the empirical nature is the high requirement of expertise. In current state of deep learning, in order to iterate ideas efficiently, both domain expertise and deep learning knowledge are required. Take medical imaging domain as an example: when designing a successful deep learning architecture for a particular medical application, the design not only needs to meet the requirements from deep learning perspective (in terms of performance, computation efficiency, and more) , but also needs to account for special properties only existed among medical images [4] (O.Ronneberger et al., 2015). Moreover, when processing medical images to be used by deep networks, many modalities require specialized image preprocessing operations so that the conventional 8-bit transformations cannot be used due to loss of information. These knowledge is often not obvious, yet crucial to the success of specific applications.

The second issue in the current state of deep learning is the inefficiency caused by manual trial and error. Figure 4 demonstrates a typical deep learning exploration workflow: given existing results, people adjust their experiment which produces new results for new adjustments. Since the manual adjustment process requires human intervention, the entire workflow can only progress forward under the presence of human appearance. As a result of this, deep learning exploration often stalls due to human's intermittent presence. For example, for a researcher who is available 8 hours a day conducting experiments each requiring 12 hours to complete, then the maximum number of experiments completed per week is 7 (one per day, due to manual experiment cycle). However, doing 7 experiments per week is only half of the maximum weekly experiment capacity, in other words, the experimentation workflow is idle 50% of the week.

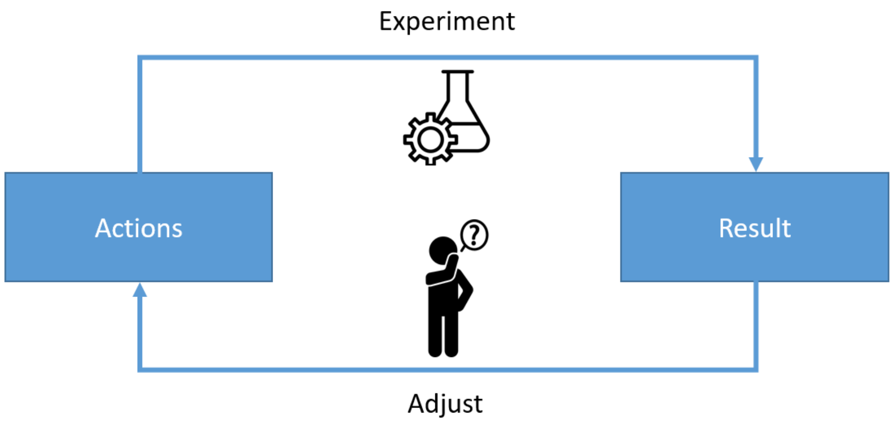

Figure 4 Manual Trial and Error Process in Deep Learning

The third issue of deep learning's empirical nature is about the difficulty of value assessment in publications. When the success of an idea heavily relies on particular empirically chosen parameters, the value of the idea will inevitably be diluted as a result. For example, Residual Network (ResNet) [5] (He et al., 2016) is one of the most impactful ideas in deep learning with more than 100k citations by the community, and it is the winner of ILSVRC challenge in 2015 with 79.7% top-1 accuracy. However, when the idea is tested alone on the same task without several empirically chosen parameters (Table

1), the top-1 accuracy would only make it to 63%. The significant difference on the results produced by the same idea, but with different empirical parameters makes it hard to assess the value of the idea itself. This also leaves room for scientific misconducts and malpractices. For example, people could abuse this by proposing an idea that does not really delivers the benefit it promises, but with justification of good results from other carefully chosen parameters.

Table 1 Parameters Important to Success of ResNet on ImageNet

Data Related	Optimization Related
• Random crop augmentation with specific parameters for padding, scaling, and aspect ratio	• Optimization rule with specific parameters (momentum, Nesterov setting)
• Inter-cubic interpolation for resizing	• Learning rate warm-up schedule
• Horizontal flip (but no vertical flip)	• Learning rate decay schedule
• Test-time center cropping	
• Specific pixel scaling with predefined mean per channel	

Among all the issues mentioned, there are solutions proposed to help resolve individual issues. For instance, requiring a more rigorous ablation study and encouraging open source practices would help address the value assessment issue. More advanced experiment design paradigms can be used to mitigate the efficiency concern. These individual solutions would certainly be helpful; however, they do not solve the overall problem fundamentally. At the same time, the dual-expertise requirement is still left as a major issue of deep learning. Fortunately, one seemingly unrelated trend in the AI community holds the potential to fundamentally overcome all three issues at the same time, and the answer is automation.

Just like other automation trends in the world, Automatic Machine Learning (AutoML) aims to automate the trial-and-error process of building machine learning/ deep learning systems. With a good automation system, it has the potential to be deployed in various domains and adapt to the field automatically without the need of expertise. At the same time, automation can also improve experimentation efficiency by automatically adjusting experiments without the need of human intervention. Furthermore, automation can be used as consistent benchmark to facilitate a fair algorithm assessment such that ideas that overly rely on specific parameters to be successful gain no additional advantage when the same tuning budget is ensured by AutoML.

AutoML trend has already started in deep learning, and has been gaining momentum every year. So far, AutoML has shown initial promises in deep learning architecture search. For example, Neural Architecture Search (NAS) [6] (Zoph et al., 2017) uses Reinforcement Learning to search for deep learning architectures and proved that machines are able to create a deep learning system as good as human. ENAS [7] (Pham et al., 2018) and DARTS [8] (Liu et al., 2019) improved the workflow efficiency of the architecture search so that the architecture search can be done within hours. The resulting architecture searched by AutoML system inspired human's choice in architecture design too, such that many later works such as [9, 10] directly leveraged these architectures for further modifications.

While AutoML has been playing an ever-increasing role in deep learning architecture design, its influence in other aspects of deep learning is much weaker in comparison. Specifically on data augmentation and deep learning optimization, people are still bothered by all three main issues mentioned earlier. More ironically, the empirical parameters listed in Table 1 all fall into either data augmentation or optimization. In other words, the research interests of AutoML are considerably weaker where we need them the most.

The current mismatch between AutoML's research interests and its real-life demands is the motivation for the dissertation research. This dissertation is an effort to fill the gap, with hope to inspire future AutoML research on domains that are overlooked by the community. This dissertation contains two major technical chapters (Chapter 3 & 4), each presenting a novel workflow to solve a particular problem in the field of optimization and data augmentation.

Chapter 3 tackles the learning rate problem in optimization from an AutoML perspective. It strives to build an automated learning rate controller (named ARC) that can monitor the training signals and can intelligently adjust learning rate towards better results and faster convergence. The key contributions of the chapter are: 1) an overall methodology for developing autonomous learning rate systems, including problem framing and dataset construction. 2) A comparison of ARC with popular learning rate schedules across multiple computer vision and language tasks. 3) An analysis of failure modes and unexpected behaviors from ARC, informing future directions for research.

Chapter 4 solves the data augmentation policy search problem with a novel AutoML solution. It proposes Random Uni-dimensional Augmentation (RUA), a method that can automatically search for optimal data augmentation policy and improve generalization performance for deep learning models. The key contributions of the chapter include: 1) A new augmentation policy search system that can achieve better results than previous best method while requiring less than half of the computation budget. 2) A new discovery of a unimodal property in augmentation parameter space. 3) Several workflow improvements that eliminate the human expertise requirement, allowing the method to be used in larger AutoML pipelines.

2 Related Work

Learning is a fundamental characteristic among most living entities. Learning can be as simple as a dog running to a food bowl after hearing the sound of cupboard opening, or it can be as complex as a PhD student doing research and writing his/her dissertation. People are particularly good at learning due to our own biological advantages [11] (Schmidt et al., 2021), but expressing how the learning happened and transferring that process to machines is complicated.

One popular approach of attempting machine learning is through introspection – analyzing human learning processes biologically and mimicking the same processes in machines through mathematical models. For example, an artificial neuron [12] (McCulloch et al., 1943) is a mathematical model to describe the behavior of a biological neuron [13] (Neuron Wikipedia, 2022) (Figure 5). It consists of inputs connected through weights (modeling synaptic connections) and the weighted sum of outcome goes through an activation function (modeling action potential), generating a non-linear output.

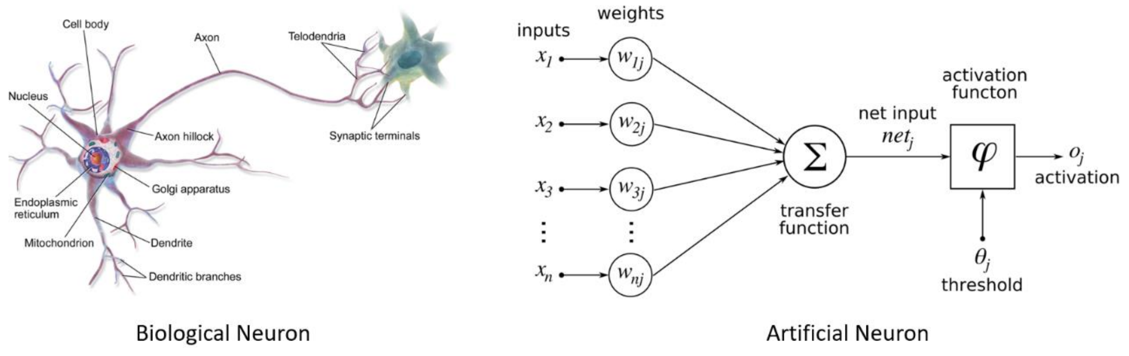

Figure 5 Biological Neuron and Artificial Neuron (From Wikipedia)

Human brain has a network of neurons consisting on the average of 86 billion neuron cells [14] (Herculano, 2012). To simulate the biological neural network, an artificial neural network combines multiple artificial neurons and forms a network of connections as shown in Figure 6. Each neuron in the artificial neural network has a pre-defined direction of information flow, and all neurons are organized as hierarchical layers. The artificial neural network learns by adjusting its internal state (also known as the weights) with respect to observations.

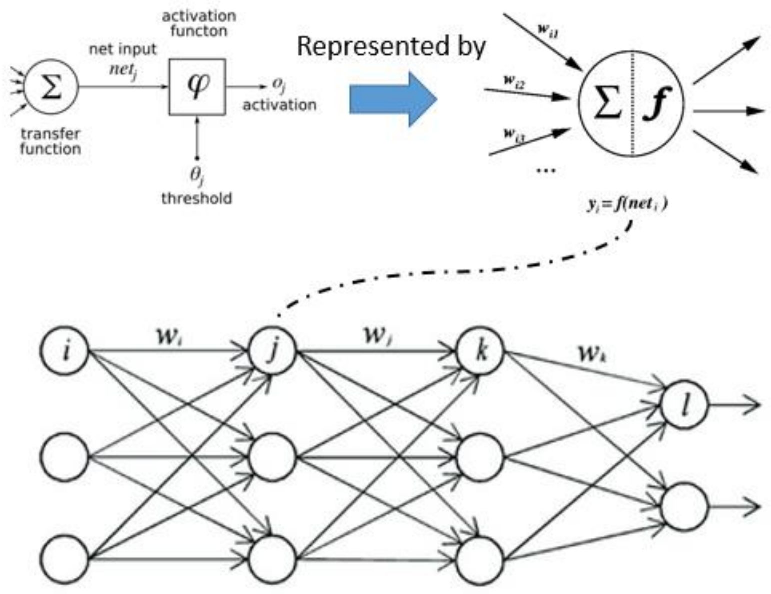

Figure 6 Artificial Neural Network

In the early days of the artificial neural network research, due to limitations on computing hardware as well as lack of efficient learning algorithms, people could only build artificial neural networks at small scale. As a result, artificial neural network was only able to perform well on simple problems and it often failed at more complicated tasks like computer vision or language understanding. One common reason for the early artificial neural network’s underwhelming performance on large scale tasks is that the network lacks the ability to express complex patterns. For this reason, researchers began adding more and more layers to the network, hoping that the expressiveness of the network would increase as the network goes deeper. Therefore, deep neural network, or deep learning became a pathway of overcoming the expressiveness concern for artificial neural networks.

The early attempts at deep learning during the 90s were not successful, because going deep in neural networks caused new issues. One of the most challenging issues is the vanishing gradient problem: the magnitude of gradient becomes smaller and smaller as error back-propagates from the end to the start. The primary cause of the vanishing gradient problem is the usage of sigmoid activation function (Figure 7). The gradient of sigmoid function is near-zero over a large input domain (> 5 and < -5), such that the final gradient would become mostly zero after several multiplications with sigmoid function’s gradient. The second challenge is about neuron’s formulation lacking consideration for structured data (like 2D image) and also having high storage complexity. For example, a typical way to process a 2D image back then was to flatten the image into one-dimensional vector, then feed it to the network. Doing this would not only remove the spatial structure in the data, but also required N^2 number of neurons, making the learning impractical in most applications.

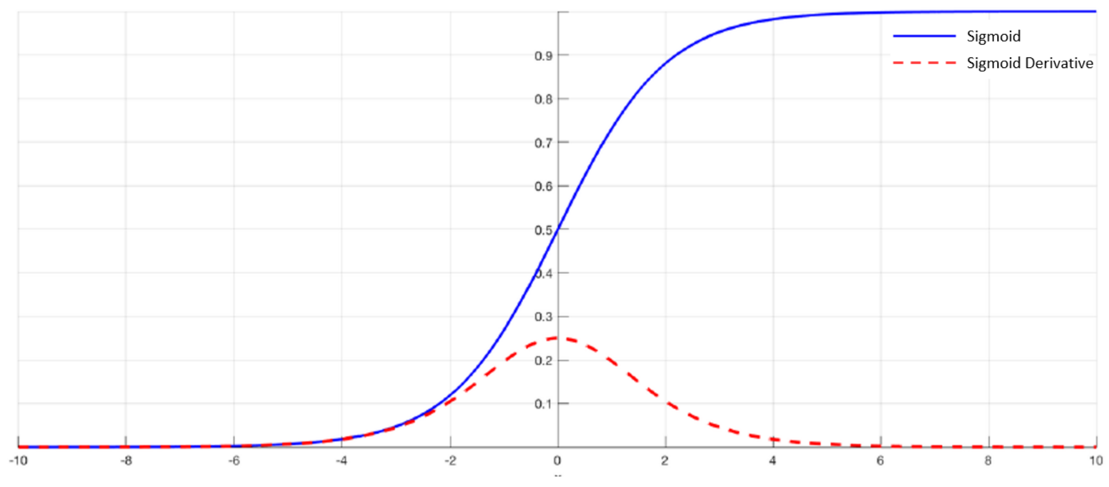

Figure 7 Sigmoid Activation Function and Derivative

Over the next two decades, breakthroughs were made on multiple fronts to finally make deep learning a reality. First, the vanishing gradient problems were overcame by the widely adopted activation functions like ReLU to prevent the gradient from vanishing, combining with smarter topology design like [5] (He et al., 2016), the magnitude of gradient can be preserved even in hundreds of layers. Spatial-aware operations like convolution achieved stunning success in a variety of computer vision tasks, also showing promise in language understanding too. The weight-sharing nature of convolution significantly reduced the number of neurons needed for computation, also allowing for massively parallel computations. At the same time, programmable GPU toolkit like CUDA made it possible for people to perform massively parallel mathematical computations directly on GPU, further speeding up the computation time significantly. Lastly, leading AI groups in industry developed open-source automatic differentiation packages like TensorFlow [15] (Abadi et al., 2016) which automated the gradient calculation and dramatically reduced the complexity of implementing a deep neural network. As a result of all these breakthroughs, deep learning start to flourish in both research community and industrial applications.

The wide adoption of deep learning has inspired the AI community to standardize its workflows and create programmable abstraction for different workflow components. For example, the majority of deep learning workflows can break into three components: 1) data pipeline 2) differentiable operation 3) optimization. Data pipeline further includes data loading, transformation, and augmentation. Differentiable operation contains gradient-based inter-network or intra-network operations. Optimization includes training loop and weight updates. The research in Chapter 3 is related to learning rate scheduling, which belongs to the field of weight updates under optimization. Chapter 4 studies data augmentation, which falls under data pipeline. Next, important concepts and related works mentioned in Chapter 3 and Chapter 4 will be briefly introduced to help understand subsequent chapters.

Learning rate is the multiplier of the update term when we perform a weight update of a deep neural network. The magnitude of learning rate represents the size of step along a particular update direction. Therefore, learning rate is also known as the step size. Given the complexity of a deep neural network's optimization surface, the learning rate needs to be properly chosen and adjusted throughout the training to ensure a good convergence.

Over the years, people have developed many empirical learning rate scheduling techniques based on different observations of optimization behavior. Learning rate warm-up [16] (Goyal et al., 2017) (Figure 8a) is a popular scheduling method that works well in modern adaptive optimizers like Adam [17] (Diederik et al., 2015) and RMSProp [18]

(Tieleman and Hinton, 2012). This is based on the observation that the adaptive update term tends to fluctuate in the beginning of the training due to the absence of prior gradients. A small learning rate value in the beginning can effectively stabilize the adaptive term and therefore help with convergence. In addition to warm-up, there are also several decaying learning rate schedulers (Figure 8 b, c) created based on other optimization behaviors. The intuition behind decaying schedules is that the learning rate needs to be large enough in the early phase of the training to ensure a wide initial exploration, then needs to gradually cool down towards the end to help with a sufficient exploitation within vicinity of local optima.

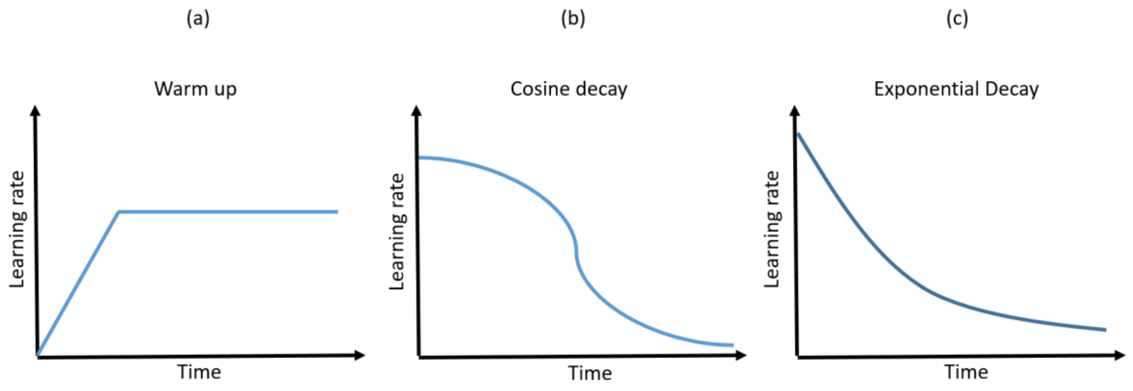

Figure 8 Different Empirical Learning Rate Schedulers

There are also more advanced learning rate schedulers that require additional learning on the target training task. For example, Super Convergence (SC) [19] (Smith and Topin, 2017) is a learning-based method that can automatically learn the learning rate schedule given any training task. Before the training starts, SC first conducts learning rate search by increasing learning rate while recording the responses on validation loss (Figure 9). Then the target learning rate is selected as the value that provides the best evaluation results (for example, accuracy). During the actual training, SC would start with learning rate warm-up, gradually increasing the learning rate to the target learning rate, then learning rate is decreased slowly until the end of the training.

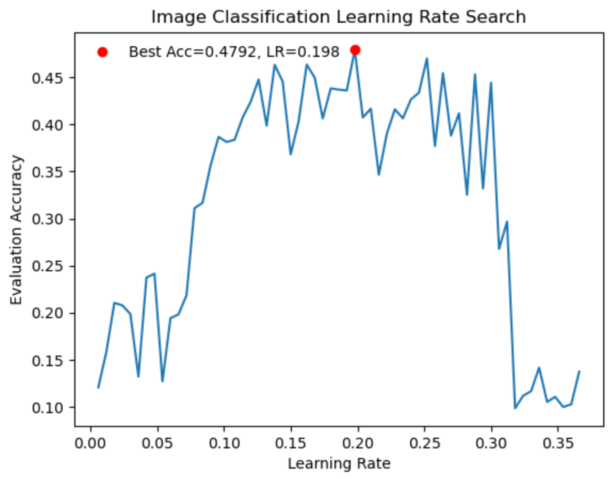

Figure 9 Super Convergence Learning Rate Search Example

For data augmentation, there are some important concepts worth clarifying and some important related works bear mentioning. Data augmentation is the practice of increasing the volume and variety of a dataset. It is a widely used technique to improve the training and generalization of a deep learning model and it has many benefits. The first major benefit of data augmentation is the increase of data volume and variety, which can help prevent over-fitting and improve generalization performance. The second major benefit is the ability to overcome certain network design limitations. For example, a notable weakness of Convolutional Neural Networks (CNNs) is rotation variance due to limitation in convolution operator. Data augmentation has been proven to become an effective approach at overcoming CNN’s rotation variance issue by applying different types of rotation to the original data (Figure 10), such that deep networks can train different kernels handling different rotations of the same object.

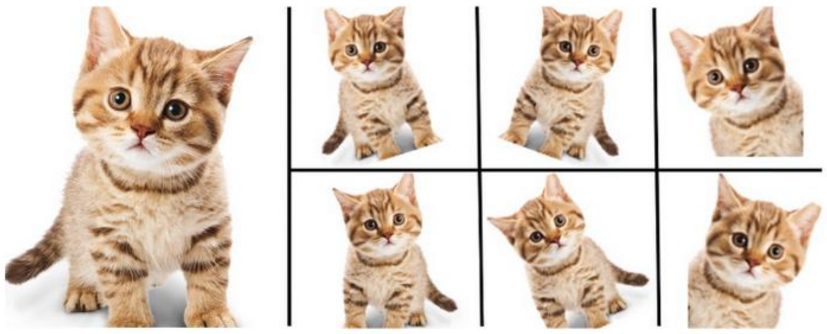

Figure 10 Data Augmentation Rotation Example (By Bharath Raj)

Creating a good data augmentation strategy typically requires human expertise and domain knowledge, which poses inconvenience when building a deep learning solution from scratch or when transferring existing data augmentation policies from other deep learning tasks. To overcome these drawbacks, AutoML community proposed several automated solutions on data augmentation.

AutoAugment [20] (Cubuk et al., 2019) and its variants FastAA [21] (Lim et al., 2019) and PBA [22] (Ho et al., 2019) automated the data augmentation process by introducing augmentation parameters which are then jointly optimized alongside the neural network parameters during training. While these methods do offer an automated solution to the problem, they also introduce massive search spaces which in turn significantly increase the time required to train a model. For example, AutoAugment uses Reinforcement Learning (RL) on a search space of size 10^{32}, which costs thousands of GPU hours to find a solution for a single task. Although later methods such as FastAA and PBA greatly improved the search and reduced computation requirements through data subsampling, they can still be undesirable due to the complexity of implementing joint optimization algorithms.

RandAugment [23] (Cubuk et al., 2020) took a different approach by completely removing the policy optimization while achieving better results than prior methods. Unlike its predecessors which rely on applying RL to a search space of size 10^{32}, RandAugment uses only two global parameters, reducing the search space from 10^{32} to 10^{2} so that a grid search can be a simple yet viable solution to the problem. As a result, RL is no longer needed for the policy search, making the method significantly easier to implement and computationally feasible for practical usage.

3 Automatic Learning Rate Controller for Deep Learning Optimization

3.1 Introduction

Learning rate is one of the most important hyper-parameters in deep learning training, a parameter that people interact with for deep learning training tasks. In order to ensure model performance and convergence speed, learning rate needs to be carefully chosen. Overly large learning rates will cause divergence, whereas small learning rates train slowly and may get trapped in a bad local minimum.

As training schemes have evolved over time they have begun to move away from a single static learning rate and into scheduled learning rates, as can be seen in a variety of state-of-the-art AI applications [24, 25, 26, 27]. Learning rate scheduling provides finer control of optimization by allowing different learning rates to be used throughout the training. However, the extra flexibility comes at a cost: these schedules bring more parameters to tune. Given this tradeoff, there are broadly two ways of approaching learning rate scheduling within the AI community.

Experts with sufficient computing resources tend to hand-craft their own learning rate schedules, because a well-customized schedule can often lead to improvements over current state-of-the-art results. For example, entries in the Dawnbench [28] (Coleman et al., 2017) are known for using carefully tuned learning rate schedules to achieve world-record convergence speeds. However, such learning rate schedules come with significant drawbacks. First, these 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 learning rate schedule often requires great expertise and significant computing resources.

In contrast, others favor existing task-independent learning rate schedules since they often provide decent performance gains with less tuning efforts. For example, some popular choices are cyclic cosine decay [29] (Loshchilov et al., 2017), exponential decay, and warm-up [16] (Goyal et al., 2017). While these learning rate 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 constant learning rate. 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, T_{multi} must all be tuned in order to function properly. Recently some methods have been proposed which either provide techniques to easily infer their parameters such as super-convergence [19] (Smith and Topin, 2017), or are even completely parameter free like stochastic line search [30, 31]. While promising, these techniques can prove finicky in practice, as will be demonstrated in the Experiment Section of this work.

Recent advancements in AutoML on architecture search [6, 7, 8] and update rule search [32] (Bello et al., 2017) 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 learning rate scheduling problem. We aim to create a system that learns how to change learning rate effectively.

Despite of several attempts that have been previously made to dynamically learn to set learning rate [33, 34, 35], these have unfortunately introduced step level dual optimization loops into the training procedure, something which can add significant complexity and training time to a problem. Moreover, this family of techniques has been shown to be vulnerable to short-horizon bias [36] (Wu et al., 2018), sacrificing long-term performance for short-term gains. We believe that both of these drawbacks can be overcome.

To that end this work introduces ARC: an Autonomous Learning Rate Controller. It takes training signals as input and is able to intelligently adjust learning rates in a real-time generalizable fashion. ARC overcomes the challenges faced by prior learning rate schedulers by encoding experiences over a variety of different training tasks, different time horizons, 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 [37] (Duchi et al., 2011)and Adam [17] (Diederik et al., 2015). Adaptive optimizers compute updates using a combination of learning rate 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 up-scaling. This is sometimes referred to as adaptive learning rate even though the learning rate term has not actually been modified. ARC is gradient agnostic and instead leverages information from various training signals to directly modify the optimizer learning rate. This allows it to detect patterns that are invisible to adaptive optimizers. Thus the two can be used in tandem for even better results. The code is publically available at [58] (FastEstimator ARC, 2022).

3.2 Challenges and Constraints of Automatic Learning Rate

Before delving into the methodology section, it is worthy of highlighting some of the key challenges in developing an autonomous learning rate controller. These constraints inform many of subsequent design decisions.

a) Subjectivity:

Determining the superiority of one model over another, each trained with a different learning rate, 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 a learning rate decision at any particular step is challenging, since the current performance is the result of the cumulative effect of all previous learning rates used during training. This is also related to short-horizon bias [36] (Wu et al., 2018), where long-term effects are easily overshadowed by short-term wins.

c) Randomness:

Randomness during training makes it difficult to compare two alternative learning rates. Some common sources of randomness are dataset shuffling, data augmentation, and random network layers such as dropout. Any meaningful performance differences due to the choice of learning rate 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 learning rate history. 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 learning rate for a given task can vary significantly, from $1e^{-6}$ up to *10* or more.

e) Footprint:

The purpose of having an automated learning rate 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.

3.3 Methodology

3.3.1 Framing the Learning Rate Control as a Learning Problem

In this work, the development of learning rate controller is framed as a supervised learning problem: predicting the next learning rate given the available training history. Due to *cumulativeness challenge*, the model needs to observe the consequence of a specific learning rate for long enough to form a clear association between learning rate and performance. Therefore, learning rate is only modified on an epoch timescale. This has a secondary benefit of dramatically reducing our computational overhead compared with competing methods.

Per challenges *subjectivity* and *scale*, as well as the desire to create a generalizable system, any task-specific metrics cannot be used. Moreover, model parameters or gradient inspections cannot be relied on either, because ARC would then become architecture dependent and would likely also fail *footprint* constraint. This work therefore leverages only the historical training loss, validation loss, and learning rate history as input features.

Due to the challenges of *randomness* and challenge *scale*, rather than generating a continuous prediction of what new learning rate values should be, this work instead poses the problem as a 3-class classification task. Given the input features, should the learning rate increase (multiply by γ, learning rate adjustment multiplier, where $\gamma > 1$), remain the same, or decrease (multiply by $1/\gamma$)?

3.3.2 Generating the Dataset

Having framed the problem, the next step is to generate a dataset on which we can train the learning rate controller model. To do this, it requires data from real deep learning training tasks with various learning rates. Therefore, the following procedure is used to generate data for each task:

1. Train n epochs with learning rate = r, then save the current state as checkpoint C
2. Reload C, train for n epochs with learning rate = $\gamma \times r$, then compute validation loss l_+
3. Reload C, train for n epochs with learning rate = r, then compute validation loss l_1

4. Reload *C*, train for *n* epochs with learning rate = $1/\gamma \times r$, then compute validation loss l_-
5. Note the learning rate which resulted in $min(l_+, l_1, l_-)$
6. Reload *C*, eliminate $max(l_+, l_1, l_-)$ and its corresponding learning rate, replace *r* with one of the two remaining learning rates at random with equal probability, and return to step 1

By executing steps 1 – 5, we can create one input/ground truth pair. The input features are the training loss, validation loss, and learning rate history during step 1, concatenated with those same features from the previous *2n* epochs of training. The label is the learning rate noted in step 5. This process is depicted in Figure 11.

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.

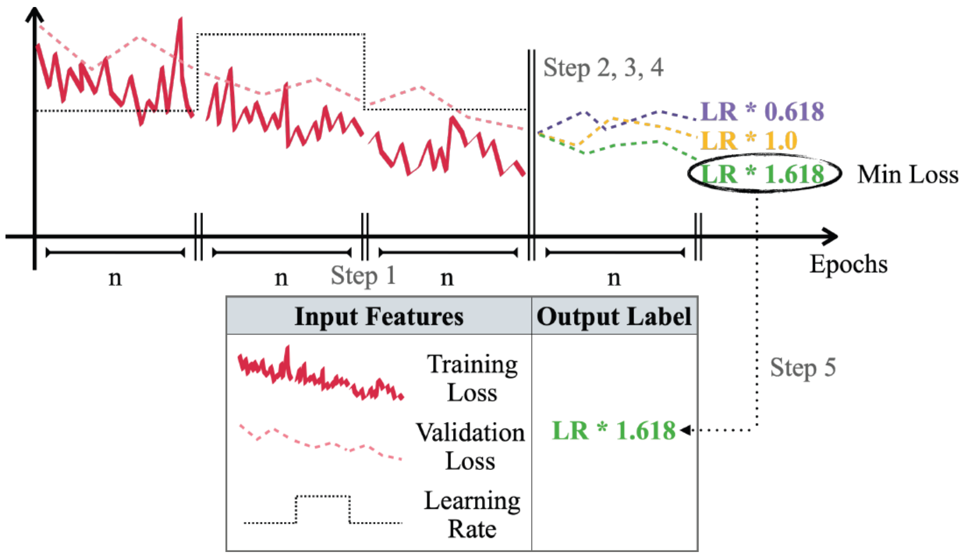

Figure 11 Workflow Illustration of Generating a Data Point When γ is 1.618

To help ensure generalization of the learning rate controller model, 12 different computer vision and language tasks are gathered, each with a configuration of dataset, architecture, and initial learning rate as shown in Table 2. Note that these tasks capture a large variety of different losses, from sparse and per-pixel cross entropy to hinge, adversarial, dice, and multi-task losses. Each of the 12 tasks were trained an average of 42 times. Each training randomly selected an optimizer from Adam, SGD, and RMSprop. The initial learning rate is selected uniformly between the minimum and maximum learning rate, *n* is selected randomly among *1* to *10*. The training of each task will last for a total of *10n* epochs. Thus approximately 5050 sample points were collected in total.

Table 2 Training Task for Data Generation

ID	Task Description	Dataset	Architecture	LRmin	LRmax
1	Image Classification	SVHN	VGG19+BN	1e-5	1e-3
2	Image Classification	SVHN	VGG16+ECC	1e-5	1e-4
3	Adversarial Training	SVHN	VGG19	1e-5	1e-3
4	Image Classification	Food101	DenseNet121	1e-5	1e-2
5	Image Classification	Food101	InceptionV3	1e-5	1e-2
6	Multi-Task Training	CUB200	ResUnet	1e-5	1e-4
7	Text Classification	IMDB	LSTM	1e-5	1e-3
8	Entity Recognition	MIT	BERT	1e-5	1e-4
9	One-shot Learning	Omniglot	SiameseNet	1e-5	1e-3
10	Text Generation	Shakespear	GRU	1e-5	1e-3
11	Semantic Segmentation	Montgomery	Unet	1e-5	1e-4
12	Semantic Segmentation	CUB200	Unet	1e-5	1e-3

3.3.3 Correcting Ground Truth

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

Luckily, there is one more data point that can be used to reduce uncertainty. Suppose that during step 6 we choose to decrease the learning rate. 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, l_-) is the same as the relative order of (l_+, l_1, 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 learning rate in determining performance. In that case we take a conservative approach and modify the ground truth label to be a constant learning rate.

3.3.4 Building the Learning Rate Control Model

For preprocessing, considering *scale* challenge, z-score normalization is applied to the training and validation losses, and then perform nearest-neighbor resizing to length 100, 200, or 300 (depending on whether we have *n*, *2n*, or *3n* epochs of history). If only *n* or *2n* epochs of prior data were available (very early during training), then zero-prepend the loss vectors to length 300. For learning rate, it is normalized by dividing by the first available value, then perform resizing followed, if necessary, by prepending ones to ensure a final length of 300.

The ARC model architecture is shown in Figure 12. It consists of three components: a feature extractor, an LSTM [38] (Hochreiter and Schmidhuber, 1997), 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 *footprint* constraint, layer parameters are chosen 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 deep learning models, this architecture should add relatively minimal overhead.

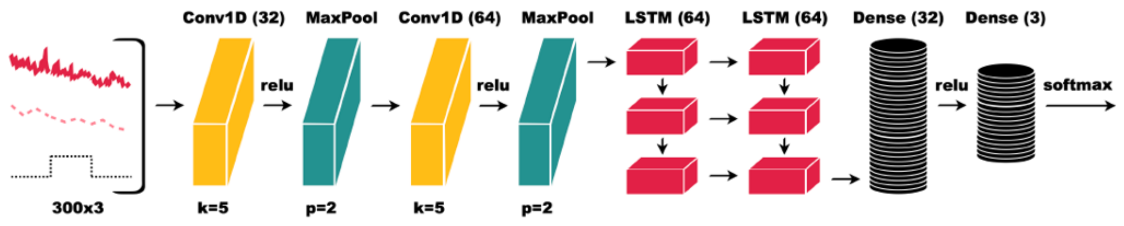

Figure 12 Learning Rate Controller Architecture

To train ARC, a hybrid loss is used which averaged classical categorical cross-entropy and a specialized binary cross-entropy. The specialized binary cross-entropy performs a one-vs.-all binarization of the task focusing on increase class, since an over-aggressive increase can lead to divergence or NaN values.

The model was trained with the corrected dataset from Section 3.3.3. Adam optimizer is leveraged to train the model with the following parameters: learning rate = $1e^{-4}$, β_1 = 0.9, and β_2 = 0.999. Training proceeded with a batch size of 128 for 300 epochs. Once trained, the ARC model can be used to periodically adjust the learning rate for other models, as will be demonstrate in Section 3.4.

3.3.5 Finding the Best Learning Rate Control Model

During the training of ARC model, it is found that the training procedure produces models with noticeably different behaviors from one another. Ideally we would select the best model based on some metric, however neither loss, accuracy, weighted accuracy, calibration error [39] (Kumar et al., 2019), nor MCC [40] (Chicco and Jurman, 2020) on held-out testing data were strongly correlated with how well the model would perform on downstream tasks (details in Section 3.5.1). Instead, a proxy problem is used to select the best model.

With proxy problem model selection, 10 different candidate models are generated by repeating Section 3.3.4 procedure 10 times. Then each candidate model is used to train an auxiliary task on multiple initial learning rates, selecting the model leading to the best average task performance across different initial learning rates for further use.

3.4 Experiments

In this section, we can see how well ARC can guide training tasks on previously unseen datasets and architectures. Specifically, ARC is deployed on two computer vision tasks (image classification on CIFAR10, object detection on MSCOCO) and one NLP task (language modeling on PTB). For each task, ARC will compare against 4 static learning rate schedules: Baseline Learning Rate (BLR), one-cycle Cosine Decay (CD), Cyclic Cosine Decay (CCD), and Exponential Decay (ED). In addition, ARC is also compared against 2 more learning based approaches: Super Convergence (SC) [19] (Smith and Topin, 2017) and SGD + Armijo Stochastic Line Search (SLS) [31] (Vaswani et al., 2019).

BLR learning rate schedule is the most basic learning rate baseline in which learning rate is held constant throughout the training. CD learning rate scheduling will decay the learning rate following half of cosine curve starting from highest learning rate in the beginning to lowest learning rate in the end. CCD learning rate schedule is CD with multiple cycles fitted within the same training. ED learning rate schedule will decay the learning rate following an exponential decaying function, with the highest learning rate in the beginning of training to lowest learning rate in the end.

As for the learning-based baselines, SC is learning-based heuristic that can automatically find the initial learning rate given any training task. Before the training starts, it first performs learning rate search by increasing learning rate while recording the evaluation response. Next it marks the learning rate with the best evaluation result as the maximum learning rate. Then SC would start the training with small learning rate and gradually increase to the maximum learning rate marked earlier, then gradually decrease the learning rate to zero. SLS is another learning-based scheduling method that learns the optimal learning rate dynamically during training. After each optimization step, SLS will perform a line search along the direction of gradient and find the learning rate that SLS decides as optimal based on mathematical conditions described in [31] (Vaswani et al., 2019).

For each task, there are 3 different initial learning rate settings used. This is because when confronted with a new dataset or network architecture, it is unclear a priori what an ideal initial learning rate will be. Ideally a learning rate schedule would therefore be robust against a variety of different initial learning rates. Therefore, in order to gain a holistic view of the effectiveness of different schedulers, 3 different initial learning rates are used. Given a task task, the maximum initial learning rate is chosen such that larger values risked divergence when training with constant learning rate, and the minimum initial learning rate is chosen such that smaller values led training with a constant learning rate to converge too slowly to be useful. Each training configuration is run 5 times, with mean test metric performance (e.g. mean test accuracy) being reported together with its standard deviation.

SC has its own process for determining which initial learning rate to use, leveraging a learning rate range test ahead of the primary training. Therefore the range test is performed for each task (range test results in Section 3.5.3). SLS computes its learning rate at every step of training, and thus does not take initial learning rate as a parameter. SLS is tested 5 times per task and report the result in comparison to the other methods for each of their initial learning rate values. The official implementation of the SLS optimizer provided by [31] (Vaswani et al., 2019) is used in the experiment.

3.4.1 Image Classification on CIFAR10

Design

For the first experiment, the training task is to perform CIFAR10 image classification [41] (Krizhevsky and Hinton, 2009). The dataset consists of 10 classes of objects (airplanes, automobile, bird, etc.), each represented by a colored image with height and width as 32 pixels. The training sample size of the dataset is 50000 and validation size is 10000.

Same architecture and preprocessing as proposed in [42] (Page, 2022) are used in the experiment, and the training ran for 30 epochs (rather than 24 in the original implementation) using an Adam optimizer and a batch size of 128. Three different initial learning rates were used: *1e-2, 1e-3,* and *1e-4*. For each initial learning rate, ARC is compared against BLR, CD, CCD (using the settings proposed in [7] (Pham et al., 2018) for CIFAR10), ED (γ = *0.9*), SC (maximum learning rate = *0.198*, detail in 2.5.3), and SLS.

The metrics used to evaluate performance of static schedulers are accuracy, convergence step. Accuracy is used to measure how model trained by each learning rate schedule can correctly classify objects. Convergence speed is defined as the number of steps needed to reach a pre-defined performance target (defined as the lowest best performance among all methods). A smaller convergence step indicates a more efficient learning rate schedule.

When comparing the learning-based baselines, to evaluate the efficiency of these advanced schedulers, two additional metrics are introduced in addition to the metrics described earlier: Overhead and Extra Training Time. The Overhead is reported in number of seconds and measures the additional time needed to operate the scheduler. The Extra Training Time is a percentage metric that measures the percentage of overhead comparing with normal training time.

Results

The results for all experiment runs are summarized in Table 3 and Table 4, with the best value highlighted in bold, and one-tail ANOVA F-statistics calculated to demonstrate statistical significance. The average-run graphs of learning rate and validation accuracy over time for each method are given in Figure 13. A summary of all runs can be found in Section 3.5.4.

Table 3 Image Classification Performance Within Static Schedules

Baselines	init_lr=0.01		init_lr =0.001		init_lr=0.0001	
	Accuracy	Convergence Step	Accuracy	Convergence Step	Accuracy	Convergence Step
BLR	90.19(0.13)	11200(683)	91.42(0.25)	10300(620)	88.82(0.09)	5500(360)
CD	**92.6(0.15)**	**6400(491)**	92.89(0.18)	7200(598)	88.22(0.11)	5600(202)
CCD	92.16(0.11)	8500(397)	92.9(0.21)	8000(362)	87.61(0.16)	7400(197)
ED	91.73(0.27)	7100(573)	92.52(0.23)	7000(451)	85.91(0.19)	9500(1190)
ARC	92(0.12)	7200(460)	**93.09(0.15)**	**6200(242)**	**91.87(0.24)**	**5200(253)**
One-Tail ANOVA F Statistics	138.57	35.02	61.41	84.06	753.96	32.68

Table 4 Image Classification Performance Within Learning-based Schedules

Baselines	init_lr=0.01			
	Accuracy	Convergence Step	Overhead (s)	Extra Training Time %
SLS	90.9(0.17)	8700(325)	202.3(6.9)	79.8%(2.7)
SC	87.6(0.26)	10800(872)	129.6(5.4)	51.1%(2.1)
ARC	**92(0.12)**	**7200(460)**	**9(0.2)**	**3.5%(0.1)**
One-Tail ANOVA F Statistics	1063.39	51.17	4111.66	1508.59

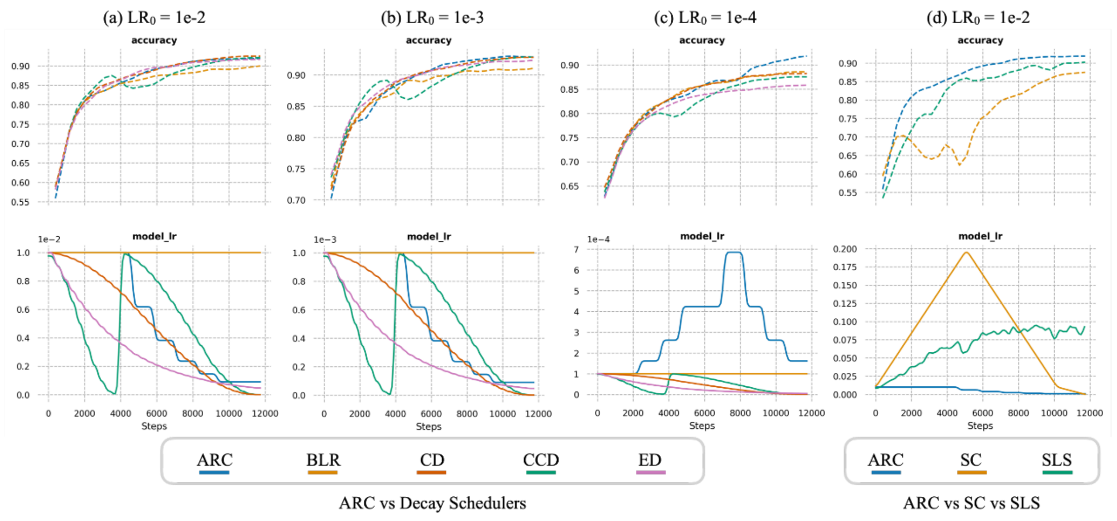

Figure 13 CIFAR10 Performance. For Better Visualization, SC and SLS are Separated From the Decay Schedulers.

Discussion

When the initial learning rate is sufficiently large (*1e*$^{-2}$ and *1e*$^{-3}$), all decaying learning rate schedulers outperform the baseline learning rate. From Figure 13(a) and (b), we can see that ARC also decided to decrease the learning rate when initial learning rate is large. Amongst all learning rate schedulers, ARC performed third best with the large initial learning rate (*1e*$^{-2}$), and was the best performer with the medium initial learning rate (*1e*$^{-3}$). Interestingly, even when ARC was outperformed by CD, it very closely emulated the CD decay pattern as shown in Figure 13 (a).

When the initial learning rate is small (*1e*$^{-4}$), however, the drawback of statically decaying learning rate schedulers becomes evident: decaying an already small learning rate damages convergence. In this case CD, CCD, and ED are all beaten by the baseline learning rate. On the other hand, as shown in Figure 13(c), ARC is able to sense that the learning rate is too small and increase it, achieving the best final accuracy. Furthermore, even given these initial learning rates, ARC outperforms every BLR configuration.

In this task, both SLS and SC require a significant amount of extra computation budget to adjust learning rates. SLS spent an average of 79.8% of extra training time to perform the line search and SC used an average of 51.1% of extra training time to learn the learning rate schedules. ARC, on the other hand, only require 3.5% of extra training time average to dynamically adjust learning rate. The extra computation needed by ARC is more than 14 times less than other methods, proving its superior efficiency over other two learning-based methods for this particular task.

3.4.2 Object Detection on MSCOCO

Design

The second and most time-consuming task is object detection. Object detection task is also known as instance detection. The goal of object detection task is to both localize the position and recognize the class of every object within the image. Therefore, the complexity of object detection is significantly higher than image classification.

The dataset used for the task is MSCOCO 2017 [60] (Lin et al., 2014), which contains 118000 training images and 5000 validation images. Each image has varying height and width, ranging from less than 100 pixels to more than 1000 pixels. The total number of classes is 80, which is also significantly more than previous task.

In order to complete the trainings within a more reasonable computational budget, the longest side of each image is downscaled to 256 pixels. The RetinaNet [43] (Lin et al., 2017) architecture was selected for this task. A batch size of 32 was used and trained for a total of 45000 steps, with validation every 1500 steps. A momentum optimizer is chosen with 0.9 for its momentum value, all other parameters consistent with the official implementation. The configuration for learning rate schedulers is the same as in Section 3.4.1, but with initial learning rates of 0.01, 0.005, and 0.001. The Super Convergence maximum learning rate was set to 0.01 (see Section 3.5.3).

The metrics used in this task is the same as in the previous task, except that, mean average precision (mAP) is used to benchmark model performance instead of accuracy. Mean average precision is a widely used metric for measuring the performance of an object detection model, with higher value being more desirable. Note that this task uses localization and focal losses which were not present in the ARC training dataset.

Results

The results for all experiment runs are summarized in Table 5 and Table 6, with standard deviation reported together with average. One-tail ANOVA test F-statistics is also provided to demonstrate statistical significance. Average-run graphs of learning rate and mAP over time for each method are given in Figure 14. A summary of all runs can be found in Section 3.5.4.

Table 5 Instance Detection Performance Within Static Schedulers

Baselines	init_lr=0.01		init_lr =0.005		init_lr=0.001	
	mAP	Convergence Step	mAP	Convergence Step	mAP	Convergence Step
BLR	16.3(0.2)	15000(835)	16.3(0.1)	23000(792)	13.4(0.1)	16700(519)
CD	16.9(0.2)	18900(962)	15.8(0.1)	26900(868)	11.4(0.1)	18200(668)
CCD	16.9(0.2)	21700(763)	15.8(0.1)	28800(702)	11.1(0.1)	25000(979)
ED	15.6(0.1)	14900(536)	14.6(0.1)	39000(558)	9.7(0.2)	36000(654)
ARC	**17.6(0.1)**	**13500(985)**	**16.7(0.2)**	**21400(983)**	**17.1(0.1)**	**12600(607)**
One-Tail ANOVA F Statistics	60.89	112.18	73.08	469.02	3159.97	1091.09

Table 6 Instance Detection Performance Within Learning-based Schedulers

Baselines	init_lr=0.001			
	mAP	Convergence Step	Overhead (s)	Extra Training Time %
SLS	9.6(0.5)	39800(1202)	13114(327.9)	78.3%(2)
SC	**17.5(0.1)**	**9800(329)**	1293.9(37)	7.7%(0.2)
ARC	17.1(0.1)	12600(607)	**471(15.2)**	**2.8%(0.1)**
One-Tail ANOVA F Statistics	1735.95	1603.48	3667.58	2998.09

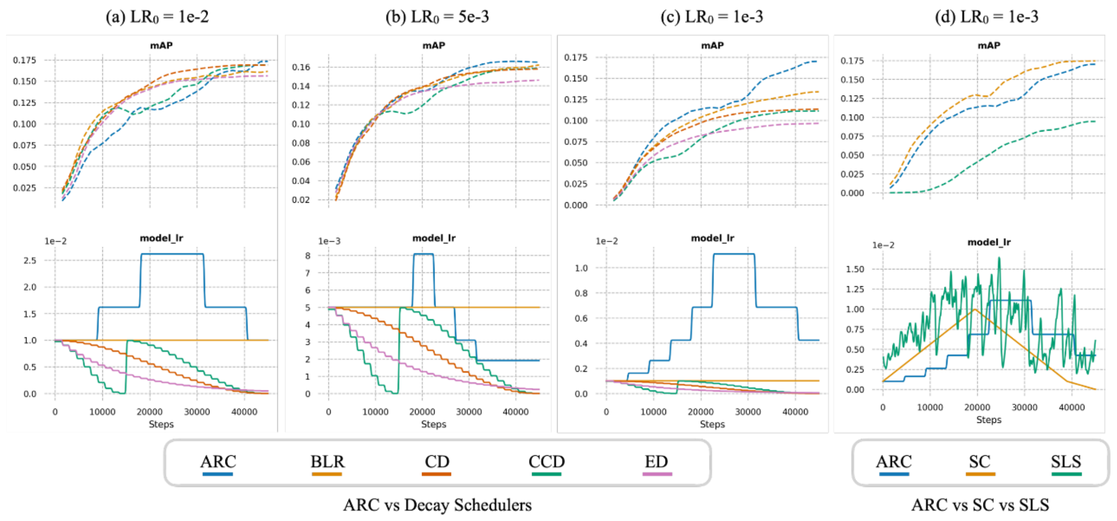

Figure 14 Average Run Performance on MSCOCO

Discussion

For instance detection task, even the largest initial learning rate (*1e*$^{-2}$) used was not large enough to allow ED to outperform the baseline learning rate. Unfortunately, any larger initial learning rates were found to lead to training divergence. This exposes a critical limitation of exponential learning rate decay: the rate of decay needs to be carefully tuned, otherwise the learning rate can be both too large early on and too small later in training. On the other hand, CD, CCD, and ARC outperform the baseline learning rate with ARC achieving the best mAP. Unfortunately SLS does very poorly on this problem despite decent performance on CIFAR.

For the other two smaller learning rates (*5e*$^{-3}$ and *1e*$^{-3}$), all of the decaying schedules are worse than the baseline learning rate because they have no mechanism to raise the learning rate and by doing so would be useful. In contrast, ARC can notice this deficiency and increase the learning rate accordingly - allowing it to achieve strong mAP across the board. SC is able to outperform ARC on this problem, but the two methods are much closer to one another than they are to any runner-up candidates. In fact, from Figure 14(d) it appears that ARC has learned to naturally mimic SC, although it is limited by the fact that it only executes once every 3 epochs.

In terms of computation efficiency, SLS spent a significant amount of computation time (78.3% extra on average) adjusting learning rate alone. In the meantime, SC has an improved efficiency (7.7% on average) comparing with image classification task (51.1% on average). Despite improved efficiency by SC, ARC spent the least amount of extra computation resource among the three learning-based methods (2.8% on average) for learning rate control, making ARC the most resource-efficient learning-based method for this task.

3.4.3 Language Modeling on PTB

Design

For the final experiment, ARC is tested beyond computer vision to verify whether it can be useful in natural language processing tasks as well. The task is to perform language modeling using the PTB dataset [44] (Marcus, 2022) with a vocabulary size of 10000, 14524 training sequences and 1152 validation sequences. Language modeling is a popular task in Natural Language Processing (NLP) with goal of predicting next word given a context. Therefore, language modeling can be used for text generation purpose. The text generation is achieved by multi-class classification of the entire vocabulary size and choosing the word corresponding to the highest probability.

The network for this problem leveraged 600 LSTM [38] (Hochreiter and Schmidhuber, 1997) 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 learning rate values of *1.0*, *0.1*, and *0.01*. The CCD scheduler used $T_0 = 14$ and $T_{multi} = 2$ such that we could fit 3 learning rate cycles into the training window. The ED scheduler γ value was set to 0.96, and the SC maximum learning rate was 23.2 (detail Section 3.5.3).

The metrics used for this task is the same as early tasks, except that perplexity is used to measure model performance. Perplexity is a metric for measuring the performance of language modeling task. Mathematically, perplexity equals to the exponent of categorical cross entropy. Therefore, a lower perplexity value means a better performance. The reason perplexity is used instead of accuracy is due to the fact that large number of classes (vocabulary size) used in language modeling tasks make it necessary to consider prediction probability overall the entire vocabulary (cross entropy) rather than only considering the prediction with the highest probability (accuracy).

Results

The results for all experiment runs are summarized in Table 7 and Table 8, with best values highlighted in bold. One-tail ANOVA F-Statistics are provided along with results to demonstrate statistical significance. Average-run graphs of learning rate and perplexity over time for each method are given in Figure 15. A summary of all runs can be found in Section 3.5.3.

Table 7 Language Modeling Performance Within Static Schedulers

Baselines	init_lr=1.0		init_lr =0.1		init_lr=0.01	
	Perplexity	Convergence Step	Perplexity	Convergence Step	Perplexity	Convergence Step
BLR	118.3(3.8)	2200(242)	136.7(2.2)	8800(103)	313.5(2.8)	8600(137)
CD	119.2(2.9)	1700(195)	157.8(2.7)	9200(206)	438.8(5.5)	8700(351)
CCD	122.6(3.5)	2700(187)	160(3.2)	16300(422)	450.1(6.8)	16100(368)
ED	122.1(2.5)	2000(258)	202(3.3)	30200(580)	603.5(3.6)	27000(449)
ARC	**111.3(0.6)**	**1600(99)**	**116.1(2.9)**	**4300(99)**	**115.9(5.2)**	**4100(73)**
One-Tail ANOVA F Statistics	20.53	12.75	633.57	2474.30	5529.31	4022.46

Table 8 Language Modeling Performance Within Learning-based Schedulers

Baselines	init_lr=1.0			
	Perplexity	Convergence Step	Overhead (s)	Extra Training Time %
SLS	150.1(7.5)	3500(395)	612.8(15.3)	63.8%(1.6)
SC	116.1(1.4)	1800(207)	437.8(14.1)	45.6%(1.5)
ARC	**111.3(0.6)**	**1600(233)**	**19.6(0.6)**	**2%(0.1)**
One-Tail ANOVA F Statistics	289.20	60.05	5437.68	1749.73

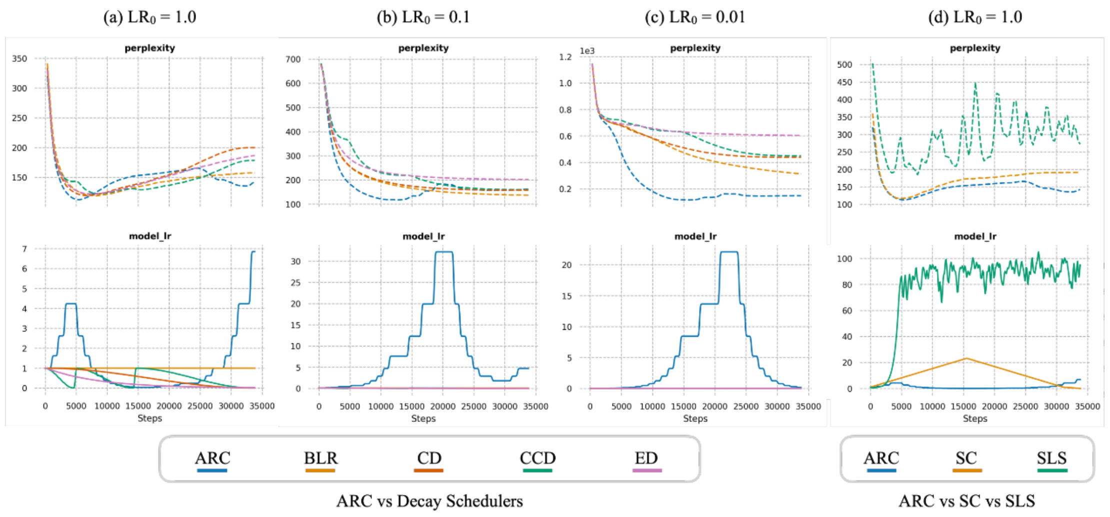

Figure 15 Average run Performance on PTB

Discussion

Surprisingly, even the largest initial learning rate did not allow CD, CCD, nor ED to outperform constant learning rate (BLR). This situation is similar to what was observed in object detection task with ED: a static decay learning rate schedule might lead to a worse performance when its decay pattern does not fit well with actual optimization needs.

In contrast, ARC shows how learning rate change can improve the training results significantly, especially when initial learning rate is small and ARC continuously increased the learning rate throughout the training, leading to the best final performance and convergence speed.

Overall in this language modeling task, ARC had very strong performance on this problem, achieving the best scores and computation efficiency regardless of which initial learning rate it was given. SC also performed well, and SLS once again struggled, though not to the same extent as for object detection.

3.5 Ablation Study and Additional Results

The goal of this section is to provide additional experimental detail mentioned in method section (3.3) and experiment section (3.4). The results for this section can be considered as supplementary results in addition to the main results. Specifically, results from Section 3.5.1 and 3.5.2 directly inform the decisions for ARC methodology design. Section 3.5.3 and 3.5.4 can help a reader better understand experiment details and results. Section 3.5.5 can help reader gain better understanding of ARC's limitations.

3.5.1 Common Performance Metrics for ARC Model Selection

Common performance metrics do not accurately predict an ARC model's downstream performance, making it difficult to determine which of several candidate models is best. During the study, 5 different candidate metrics are considered and tested, along with the final models from each run, and the proxy task from Section 3.5.2.

The candidate metrics considered were accuracy, MCC, validation loss, calibration error, and weighted accuracy (specified in Equation 1 and Table 9).

$$W_{acc} = \frac{\sum_i CM\,[i,i] \times |\,RPM[i,i]|}{\sum_{i,j} CM\,[i,j] \times |\,RPM[i,j]|} \qquad (1)$$

In Equation (1), the CM is the confusion matrix and RPM is the Reward Penalty Matrix (Table 9).

Table 9 Reward Penalty Matrix

Predict/Actual	Decrease	Constant	Increase
Decrease	+3	-1	-3
Constant	-1	+1	-3
Increase	-3	-1	+3

To test whether any of these metrics were useful, the ARC training procedure was run a total of 5 times. During those runs, the best model was saved according to each of the different metrics, such that in the end there are 5 ARC models for each metric, where models may or may not be the same across metrics (it could be that for one training run, the model with the best accuracy also had the highest MCC).

After getting the best model selected by each metric, it will be used to train a 9-layer residual network for 30 epochs on CIFAR10, recording the best evaluation accuracy during training. As always, three different values of initial learning rate: 0.01, 0.001, and 0.0001 are tested, where each training is repeated 5 times. The results are summarized in Figure 16.

Unfortunately, it seems that none of the simple metrics are strongly indicative of ARC model effectiveness. In fact, simply taking the model from the end of training (epoch 300) was just as effective as any of the metrics we tested. This motivated the use of a proxy problem.

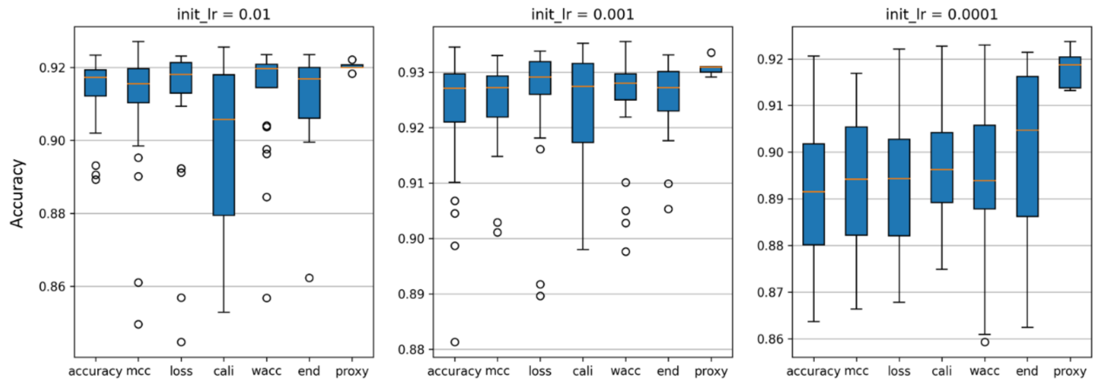

Figure 16 Evaluating Models Selected by different Metrics

3.5.2 Model Selection via a Proxy Task

Since it is found that metrics considered are unable to predict ARC performance reliably, a proxy task is considered for model selection instead. First 10 candidate ARC models are trained to the end without any metric. Then each model will be evaluated on a proxy test: WideResnet28 architecture on the SVHN Cropped dataset for 30 epochs, repeating each training 5 times.

Similar to metric testing, 3 different values of initial learning rates: 0.1, 0.001, and 0.00001 are used for proxy test. The best accuracies from each of these SVHN training runs are gathered, and the best candidate model is selected with the best mean. Results for each candidate are given in Figure 17. Notice that candidates 1 and 7, which tended to diverge at high learning rates, are naturally eliminated by this process. Model 4 was selected for later use.

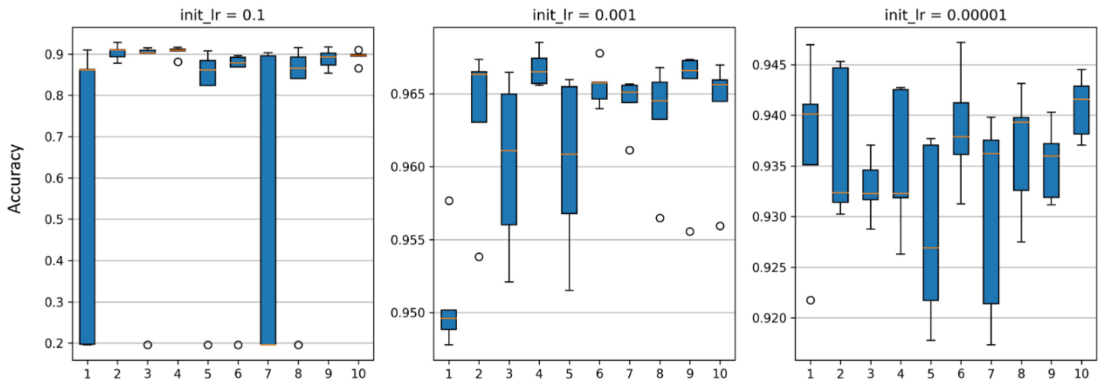

Figure 17 ARC Candidate Performance on Proxy Problems

3.5.3 Super Convergence Learning Rate Search

To use Super Convergence learning rate scheduling, one first needs to run a search routine to determine the ideal maximum learning rate. For CIFAR10, the learning rate routine is the same as found in the official paper [19] (Smith and Topin, 2017): learning rate is increased linearly from 0.0 to 3.0 over 5000 iterations, and evaluation accuracy and learning rate are plotted accordingly. The maximum learning rate is the one that leads to the maximum accuracy (Figure 18).

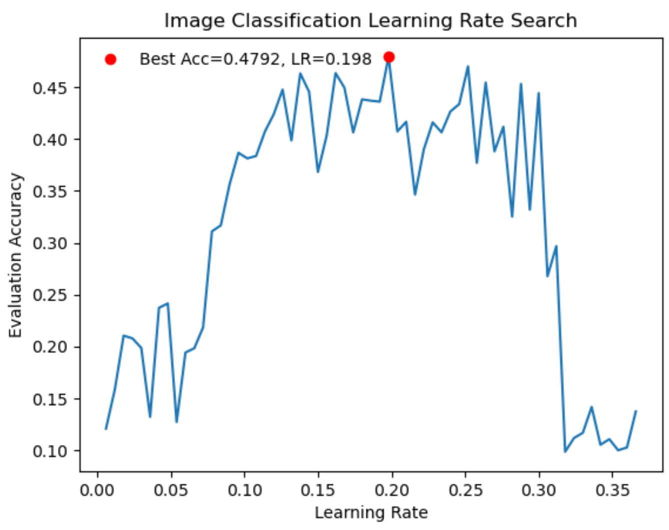

Figure 18 Super Convergence Learning Rate Search for CIFAR10

The learning rate search configuration for other tasks is the same, except that for instance detection learning rate is increased linearly from 0.0 to 1.0 (Figure 19), and for language modeling learning rate is increased from 0.0 to 100 (Figure 20). For instance detection, the evaluation loss is used rather than mAP because the magnitude of the latter is extremely small during early stages of training. As a consequence, one should look for the minimum metric score rather than the maximum when searching for the appropriate max learning rate in this task.

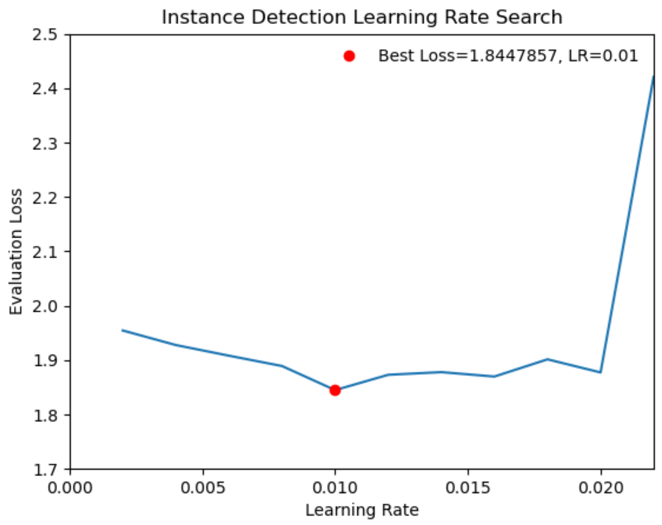

Figure 19 Super Convergence Learning Rate Search for MSCOCO

After finding the maximum learning rate for all 3 tasks, the initial learning rate is set to be $\frac{\max learning\ rate}{f}$, where f is selected between 5 to 40 (which is consistent with the original paper's experiment section). The f chosen for image classification, instance detection, and language modeling are 19.8, 10.0, and 23.2 respectively, such that their initial learning rate values match those used in other schedulers.

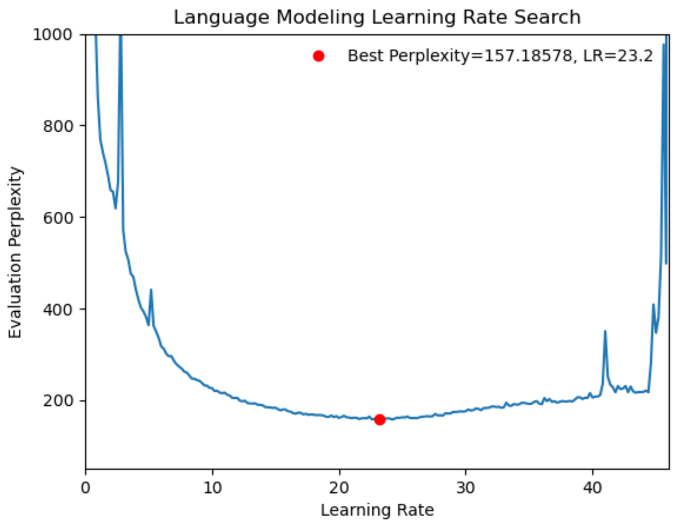

Figure 20 Super Convergence Learning Rate Search for PTB

3.5.4 Performance Summary for all Experiments

In this section, the full performance metrics (accuracy, mAP, perplexity) summaries for all experiments from Section 3.4 is presented. Each box in Figures 21, Figure 22, and Figure 23 contain 5 data points from each of 5 independent runs. SC is only visualized for the initial learning rate indicated by the learning rate range test for the particular task (Section 3.5.3).

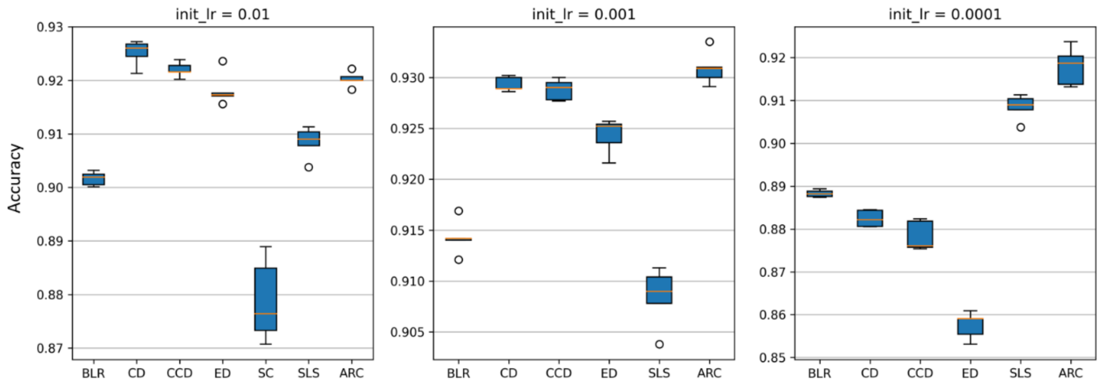

Figure 21 Image Classification Accuracy Summary for All Runs

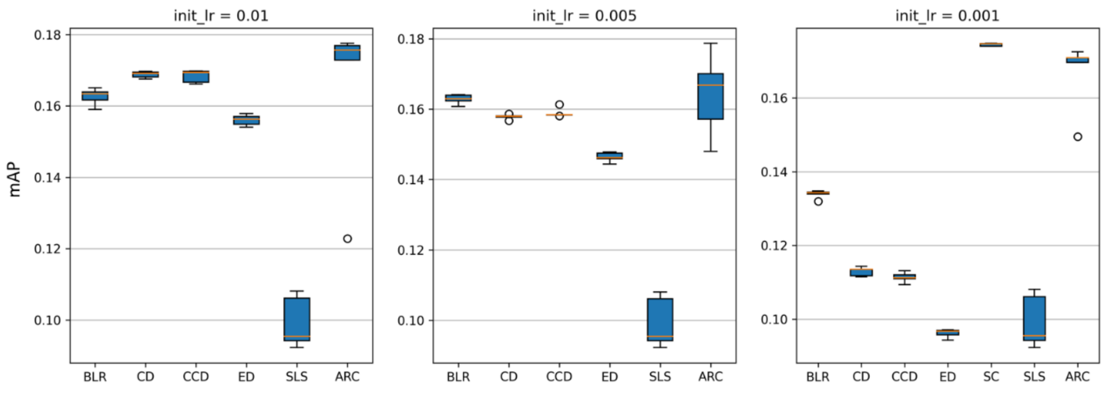

Figure 22 Instance Detection mAP Summary for All Runs

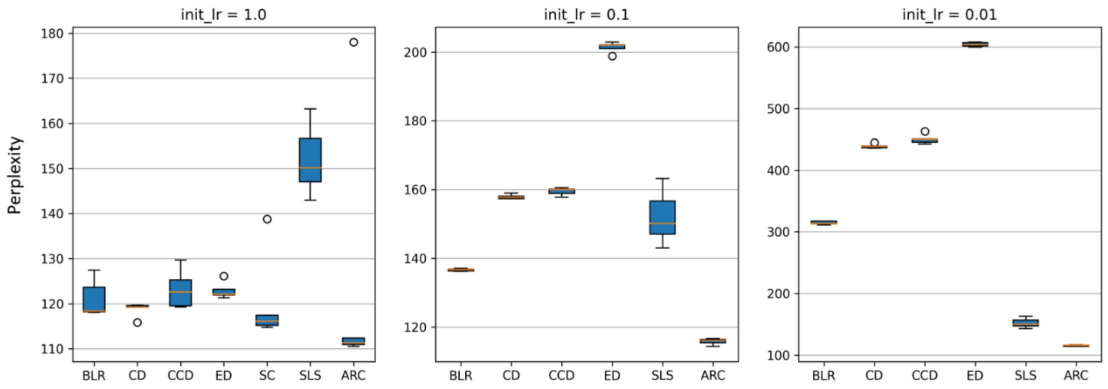

Figure 23 Language Modeling Perplexity Summary for All Runs

Note that there is an extreme outlier for ARC in PTB language modeling with initial learning rate = 1.0 in Figure 23. As Figure 24 demonstrates, this outlier is a result of an abnormal initialization (run 1 in Figure 24), with initial perplexity being more than two times higher than other runs.

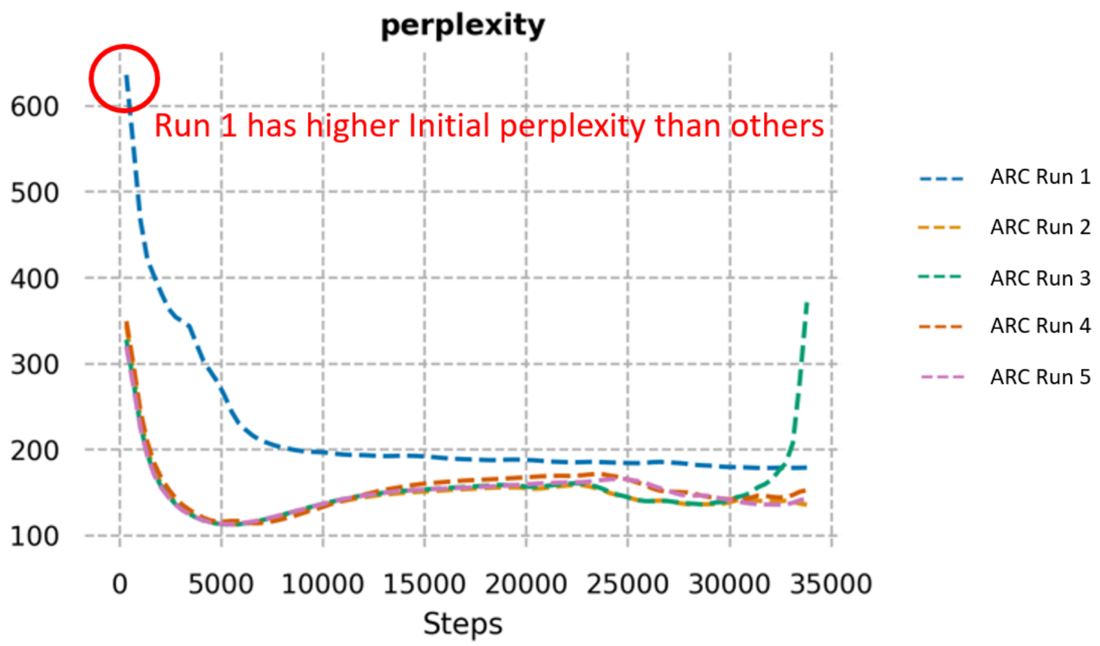

Figure 24 ARC runs on PTB with Large Initial Learning Rate

In addition, the convergence speed for all summaries is presented in Figure 25, Figure 26, and Figure 27. Note that the convergence speed for SLS and SC are only available for the initial learning rate searched by SC.

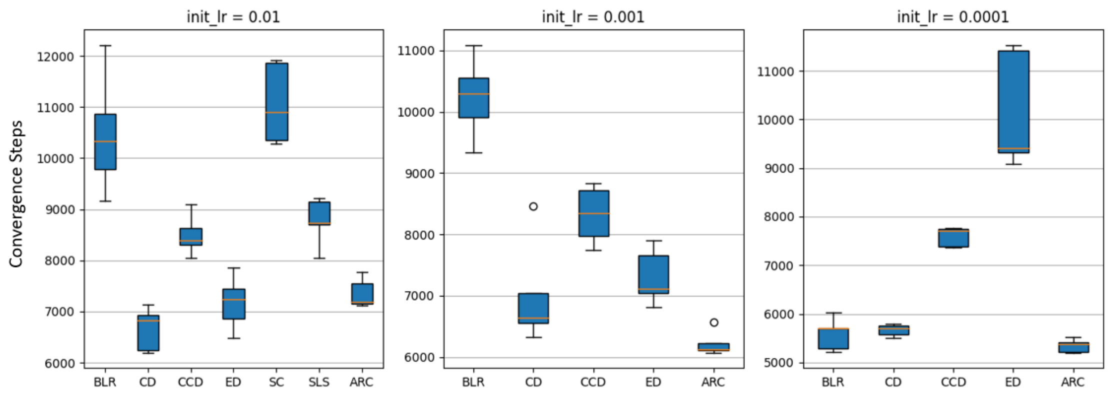

Figure 25 Image Classification Convergence Speed Summary for All Runs

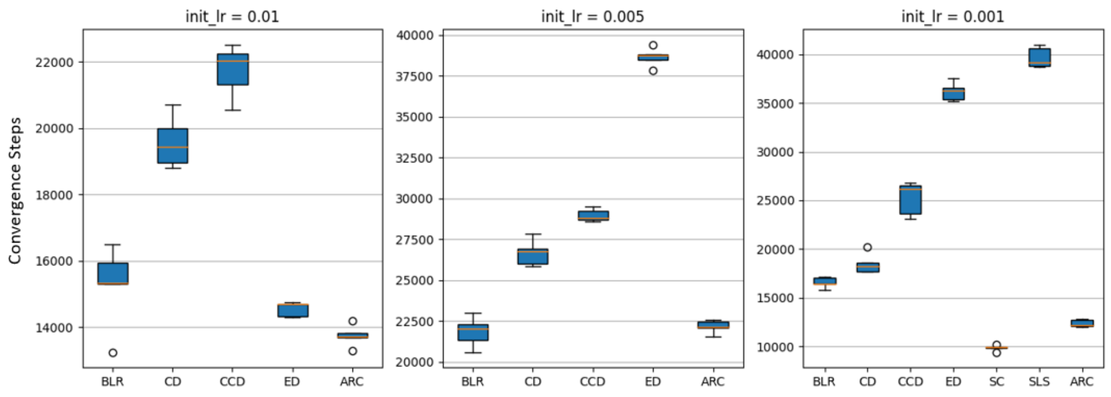

Figure 26 Instance Detection Convergence Speed Summary for All Runs

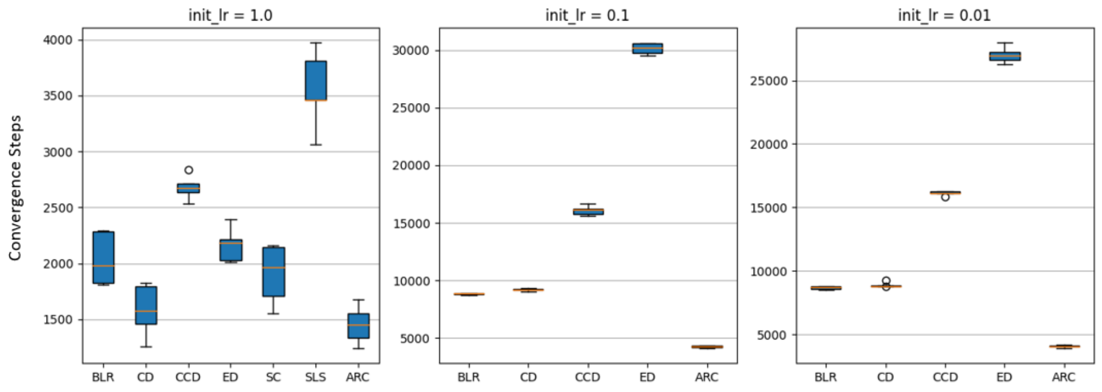

Figure 27 Language Modeling Convergence Speed Summary for All Runs

3.5.5 ARC Invocation Frequency

ARC executes at the per-epoch rather than per-step timescale, but this leaves the question of how many epochs should elapse in between invocations. An easy answer would be 'as often as possible', but invocations which are too frequent may lead to greater instability or vulnerability to short-horizon bias.

To find an ideal frequency, 5 ARC models are trained and then deployed on top of a 9 layer residual network to train on the CIFAR10 dataset at initial learning rate values of 0.01, 0.001, and 0.0001. For each initial learning rate, different invocation frequency n (from 1 to 6) are tested. Every experimental configuration was repeated 5 times, with maximum accuracies summarized in Figure 28.

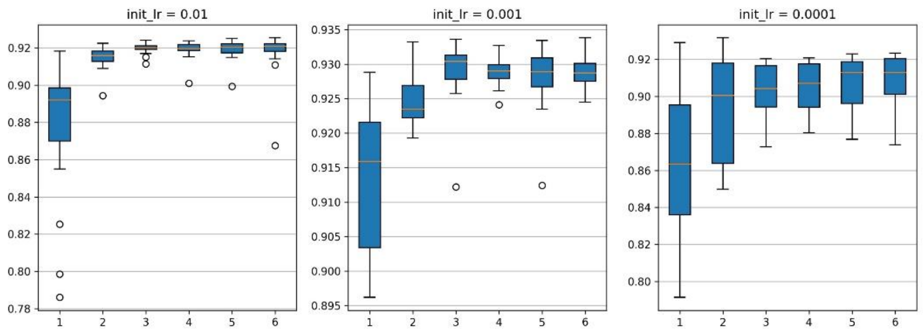

Figure 28 Effect of ARC frequency on Final Performance

For *n* in [1, 3], the training proceeded for 30 epochs. For *n* in [4, 6] the training ran for *10n* epochs such that each configuration would have an equal opportunity to adjust the learning rate. From the results, it can be observed that once every 3 epochs is the most frequent invocation schedule we can currently support without harming performance.

3.6 Limitations and Unexpected Behaviors

As Section 3.4 demonstrates, ARC can be successfully deployed over a range of tasks, architectures, optimizers, dataset and batch sizes, and initial learning rates. 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. It is observed that once every 3 epochs is the best frequency (see Section 3.5.5). The need for 3 epochs may be attributable to the way in which we train ARC using short-, mid-, and long-term data, or else perhaps a consequence of short-horizon bias [36] (Wu et al., 2018). Notably, in the only instance where ARC performed worse than SC, it was essentially emulating SC but was unable to do so quickly enough. See Figure 14(d). It may be possible to avoid this limitation by measuring validation loss more frequently, rather than once at the end of each epoch.

As for failure modes, just like any other deep learning model, ARC can also make incorrect decisions. For example, when training MSCOCO at an initial learning rate of 0.01, 4 out of 5 ARC trainings noticeably outperformed all competing schedules. There was, however, an outlier which performed significantly worse. Its training plot is shown in Figure 29(a) alongside with two other ARC training runs. In all 3 of the visualized training runs, ARC raised the learning rate too aggressively (circled in red in the Figure). In the blue and green runs, ARC automatically detected that the learning rate was too high and decreased it aggressively, leading to strong final performance. In the orange (failure) case, ARC for some reason doubled down on the excessively large learning rate, leading to further degradation in performance.

Although this is clearly undesirable behavior, it is quite rare - occurring only three times across our total of 45 different experimental configurations/runs (one time in each of the different PTB initial learning rate configurations). Since real-world applications tend to train multiple models and keep the best one, we do not foresee this being usage-limiting.

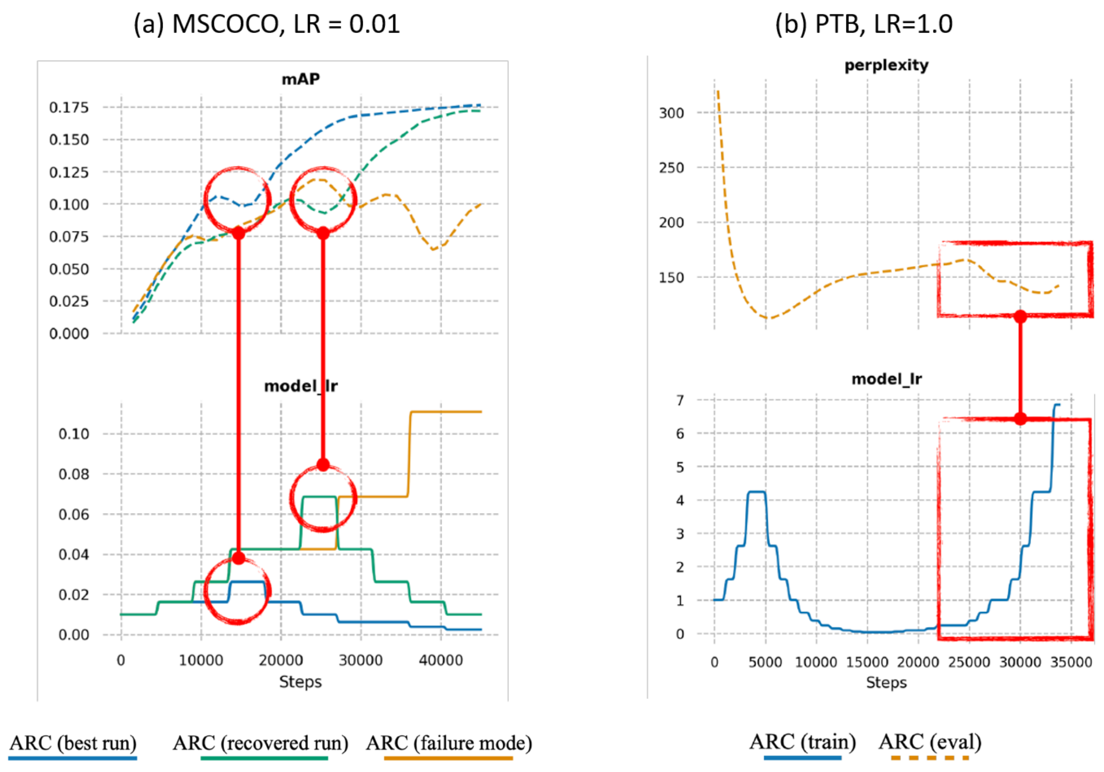

Figure 29 Failure Modes and Unexpected Behaviors

Figure 29(b) shows an interesting phenomenon observed in language modeling training which is unexpected. After achieving an optimal performance around step 5000, ARC started to drop the learning rate as might normally be expected to improve performance. However, after step 25000 it changed course and dramatically increased the learning rate. Comparing ARC's performance with the other schedulers in Figure 15(a), it seems that ARC may be attempting to prevent the model from over-fitting.

3.7 Chapter Conclusion

This chapter proposed an autonomous learning rate controller that can guide deep learning training to reliably better results. ARC overcomes several challenges in learning rate scheduling and is complementary to modern adaptive optimizers. ARC’s superiority to popular schedulers is experimentally demonstrated across a variety of tasks, optimizers, batch sizes, and network architectures.

In addition, ARC achieves its objectives without tangibly increasing the training budget, adding additional optimization loops, nor introducing complex RL workflows. This is in sharp contrast to prior work in this field, as well as other AutoML paradigms in general. 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 on the part of the end user makes it a valuable addition to the AutoML domain.

4 Practical Automatic Data Augmentation System Design

4.1 Introduction

Data augmentation is a process of increasing the volume and variety of datasets through various transformations. It is a widely used technique in computer vision to improve deep-learning model's generalization performance. Data augmentation has many benefits; it can overcome some intrinsic limitations of convolutional neural networks (e.g, rotation variant). In addition, data augmentation can also help prevent over-fitting due to the increase of data volumes and data variety such that the network is less likely to be fixated on high-frequency noises or nonessential patterns.

Data augmentation is also a widely used approach to improve generalization performance [24] (Bochkovskiy et al., 2020) in a real-world application scenario because it can improve model performance metrics without incurring additional computation costs at inference time. Unfortunately, creating a good data augmentation strategy typically requires human expertise and domain knowledge [23] (Cubuk et al., 2020), which is inconvenient during initial development as well as when transferring existing strategies between different tasks. In an effort to over-come these drawbacks, researchers have begun looking for an automated solution to data augmentation.

Automated data augmentation at its core can be viewed as a distribution-matching workflow. A deep learning model trained on a specific training set is expected to generalize well if the distribution of training data matches or includes the distribution of testing data. However, in reality where such assumption about distribution is not guaranteed, one has to rely on data augmentation to either extend/shift the distribution of training data towards testing data (Figure 30). For example, a common way to introduce rotation-invariance in Convolutional Neural Networks (CNNs) is to rotate the data randomly during training so the final model can be robust against different possible rotations.

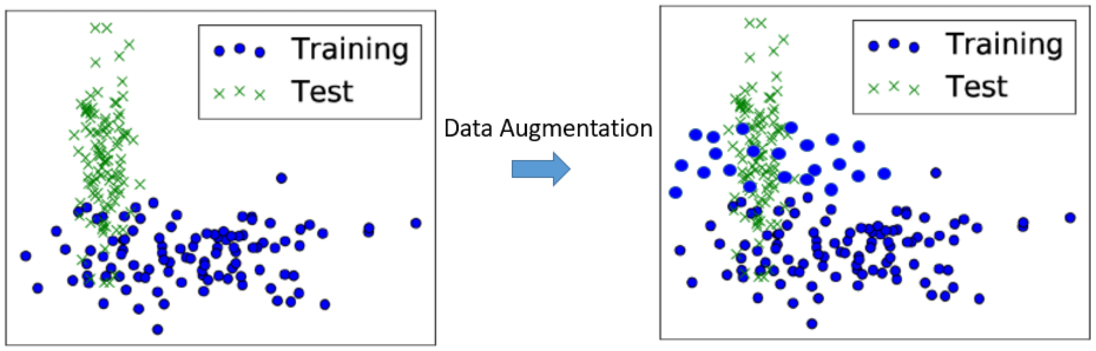

Figure 30 Viewing Data Augmentation as Distribution Matching

Since expanding the training data's distribution can help improve generalization at testing time, it might be tempting to extend the training data's distribution infinitely to guarantee the coverage of testing distribution. However, over-expanding training data's distribution often lead to worse generalization performance in real-life applications. The primary reason is that as we are expanding the distribution of the training set, the training task becomes significantly harder to learn. As a result, it often requires a much more complex network architecture and significantly longer training time in order to reach the same training error. Given the same network and computing time, the training performance usually becomes worse.

Therefore, given the trade-off between under-expanding and over-expanding the training data distribution, search of optimal data augmentation policy for a given task in an automated way becomes the pursuit of many research works. A good automated augmentation workflow can improve a deep learning model's robustness to unseen data and improve testing performance efficiently.

AutoAugment [20] (Cubuk et al., 2019) automated the data augmentation process by introducing augmentation parameters which are jointly optimized alongside the neural network during training. It frames the problem as a dual-optimization problem (Figure 31). The iterative process consists of two loop hierarchies: the inner loop is the normal training of target neural networks, the outer loop is the training of a separate policy selection Recurrent Neural Network (RNN) model. Since the augmentation operations are not easily differentiable, there is no gradient between the augmentation policy and validation result. Because of this, Reinforcement Learning (RL) is used to train the augmentation agent RNN instead.

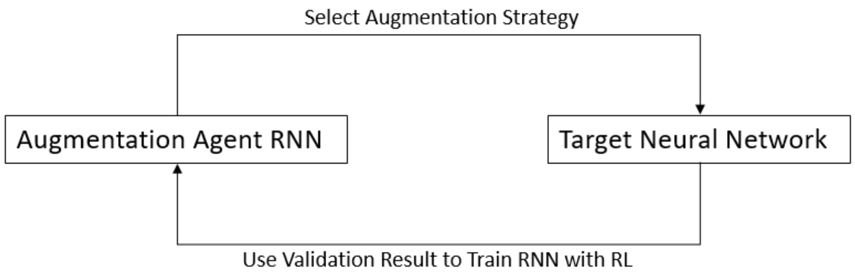
Select Augmentation Strategy
Augmentation Agent RNN
Target Neural Network
Use Validation Result to Train RNN with RL

Figure 31 AutoAugment Dual Optimization Workflow

The search space of AutoAugment include 16 operations: Shear-X, Shear-Y, Translate-X, Translate-Y, Rotate, AutoContrast, Invert, Equalize, Solarize, Posterize, Contrast, Color, Brightness, Sharpness, Cutout, and Sample Pairing. Each operation further contains 10 different magnitudes and 11 different probabilities of applying the operation. As a result, the overall search space of using 5 sub-operations is in the order of 10^{32}. Given the order of the search space, RL is used to search for an optimal solution among the policy. It uses validation accuracy as reward to improve the RNN model.

The AutoAugment method is one of the earliest attempts to tackle the automatic augmentation problem, and it lays the foundation for many future works. One of the key contributions is the idea of quantifying the augmentation space and perform optimization in the space. Before that, data augmentation is mainly viewed as an independent and discrete operator and often needs to be tuned manually through human expertise. Another contribution of AutoAugment is that it is the first study that showed optimizing data augmentation policy can improve generalization performance and achieve state-of-the-art result. These two contributions inspired many follow-up research studies.

On the other hand, AutoAugment has lots of room for improvement too. First, they introduced massive search spaces which significantly increases the time required to train a model from a few hours to thousands of hours. Another weakness is the complexity of workflow, because it requires two different hierarchies of optimization. Implementing complex RL optimization on top of deep learning training makes AutoAugment undesirable in many real-world applications.

Population Based Augmentation (PBA) [22] (Ho et al., 2019) is a method built on AutoAugment that alleviates the need for a massive computation resource. It generates non-stationary augmentation policy schedules instead of a fixed augmentation policy. Through their formulation of the search, the search becomes more efficient despite the fact that the search space is increased. In the end, they achieved similar performance as AutoAugment method while using significantly less GPU time overall and also reduced the complexity of implementing the workflow.

Fast AutoAugment (FastAA) [21] (Lim et al., 2019) pushes the concept of distribution matching to a new level by maximizing the match between distribution of augmented data and un-augmented data. It learns the augmentation policy that considers the data after augmentation as missing data points among training data. FastAA also replaced the RL agent with a distribution-matching agent which is much easier to train than RL algorithms. They managed to reach equivalent performance as AutoAugment and PBA while requiring both less computation and less complexity of implementation.

RandAugment [23] (Cubuk et al., 2020) is an automated system that can help find the best augmentation policy for a given deep learning task. Unlike its predecessors which rely on RL or evolution-based methods to large search space, RandAugment adjusts only two global parameters, reducing the search space from 10^{32} to 10^2. As a result, RL is no longer needed for the policy search, making the method significantly easier to implement and more computationally feasible for practical usage.

There are 14 augmentation operations used by RandAugment: Identity, AutoContrast, Equalize, Rotate, Solarize, Posterize, Color, Contrast, Brightness, Sharpness, Shear-X, Shear-Y, Translate-X, and Translate Y. All operations above are controlled by two global parameters – M and N. M represents the distortion magnitude which controls the intensity of each transformation. At each training step, N transformations will be drawn randomly (uniform probability, with replacement) from the 14 available options and applied to the training data. By default, M and N are both integers ranging from 1 to 10, with 10 giving the maximum augmentation effects. However, one can always configure their maximum to be higher than 10 to further increase distortion effects or to create a finer resolution in the search space.

Despite the significant complexity and efficiency enhancements made by RandAugment, there is still room for improvement. For example, a default 10 by10 grid search suggested by RandAugment requires repeating the training 100 times to find the best augmentation settings. This may easily exceed many real world training budgets.

To reduce costs, a sub-grid is often selected from the 10x10 grid for the actual search. Unfortunately, appropriate sub-grid selection is highly customized to specific problems. This re-introduces a requirement on human expertise and experience, which autonomous methods seek to avoid. For example, for CIFAR 100 [41] (Krizhevsky and Hinton, 2009) the proposed sub-grid is $N \in \{1, 2\}, M \in \{2, 6, 10, 14\}$ For ImageNet [45] (Deng et al., 2009), a ResNet50 model [5] (He et al., 2016) uses the sub-grid: $N \in \{1, 2, 3\}, M \in \{5, 7, 9, 11, 13, 15\}$, whereas EfficientNet [9] (Tan et al., 2019) on the same dataset searches $N \in \{2, 3\}, M \in \{17, 25, 28, 31\}$. It is difficult to say what kind of intuition would allow someone to generate such sub-grids for previously unseen problems.

To address these problems, this chapter proposes Random Uni-dimensional Augmentation (RUA): a simpler yet more effective automated data augmentation workflow. The goal of RUA is to achieve the following two objectives: (1) Reduce the computational cost required to perform automated search, without sacrificing performance. (2) Eliminate the need for problem-specific human expertise in the process, enabling a fully automated workflow. The code is publically available at [59] (FastEstimator RUA, 2022).

4.2 Methodology

4.2.1 Dimensionality Reduction: 2D to 1D

There are 2 global parameters defined in the search space of RandAugment: *M* and *N*. *M* represents the global distortion magnitude which controls the intensity of all augmentation operations. *N* is the number of transformations to be applied in each training step. By default, *M* and *N* are both integers ranging from 1 to 10, with 10 giving the maximum augmentation effects.

Although the definitions of *M* and *N* are different, the end result of increasing their values is the same: more augmentation. If they could be merged into a single augmentation parameter, then the search space could be reduced by an order of magnitude. To check whether this might be possible, RandAugment is performed on a full *10 x 10* grid for two classification tasks: ResNet9 for CIFAR10, and WRN-28-2 [46] (Zagoruyko and Komodakis, 2016) for SVHN [47] (Netzer et al., 2011). Their test accuracies are shown in Figure 32.

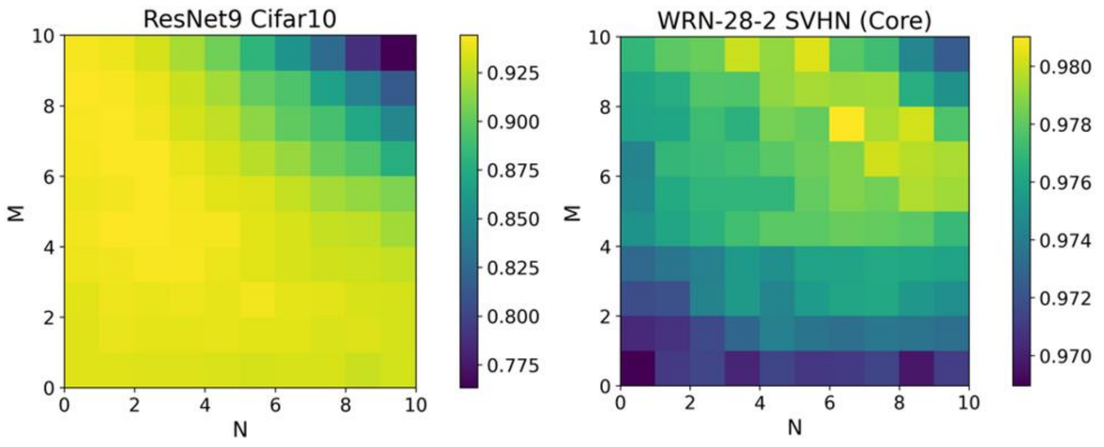

Figure 32 Model Accuracy as Function of M and N Using RandAugment

The color gradients in Figure 32 show a diagonal trend from the bottom left to the top right. Although the optimal accuracy regions vary between the two problems, the fact that both exhibit an approximately diagonal gradient raises the possibility of traversing the two parameters simultaneously. It is also confirmed that this pattern is robust to variations in augmentation parameters, as well as across different architectures, tasks, and datasets. These results can be found later in section 4.2.4.

To make use of the color gradient, we can introduce a single parameter $r \in \{0, 1\}$ such that $r = \frac{M}{M_{max}}$ $and\ r = \frac{N}{N_{max}}$. As a result, our augmentation operation parameters can be directly defined in terms of r, eliminating the need to pick an explicit value for M_{max}. This parameterization can be later found in Table 10 from section 4.2.3. This formulation leaves N_{max} as the single open parameter in the method. While one could simply set N_{max} as 10, in the footsteps of RandAugment, it can also be set lower while still providing adequate gradient traversal. We defer further discussion of this to Section 4.2.4.

In situations where $r \times N_{max}$ is not an integer, a floor operation is applied on $r \times N_{max}$ to decide the number augmentations, plus a final augmentation which executes with probability equal to the floating point remainder. For example, if $r \times N_{max} = 3.14$, then 3 augmentations will be guaranteed, and a fourth will execute with 14% probability.

4.2.2 More Search with Less Computation

Another interesting observation one can make from Figure 32 is that, traversing the diagonals of both CIFAR 10 and SVHN, accuracy first increases to a maximum and then decreases. In other words, there appears to be unimodality with respect to r. If we extract these diagonal terms and plot their relative accuracies against *r* the unimodal trend becomes more apparent as shown in Figure 33.

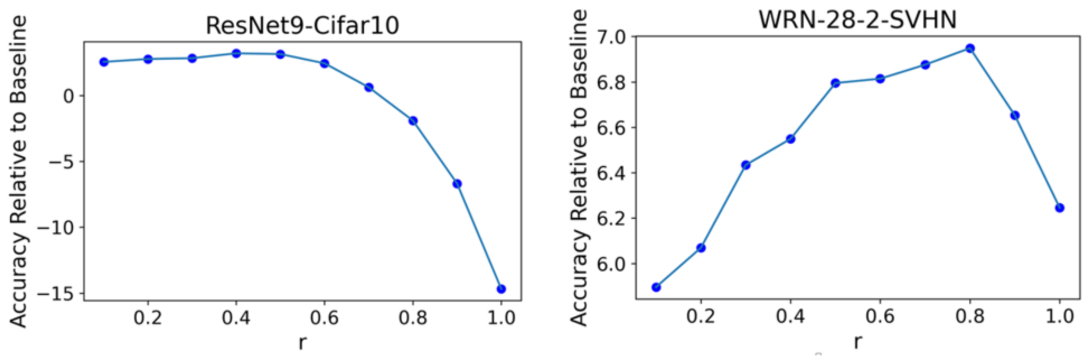

Figure 33 RandAugment Test Accuracy as Function of r (Nmax = 10)

To further verify this phenomena, a different architecture is used and applied to RUA transformations from Table 10 (right), then plot the relative performance against *r* in Figure 34. As we can see, despite some randomness on the local scale, the overall unimodal relation between accuracy and *r* persists.

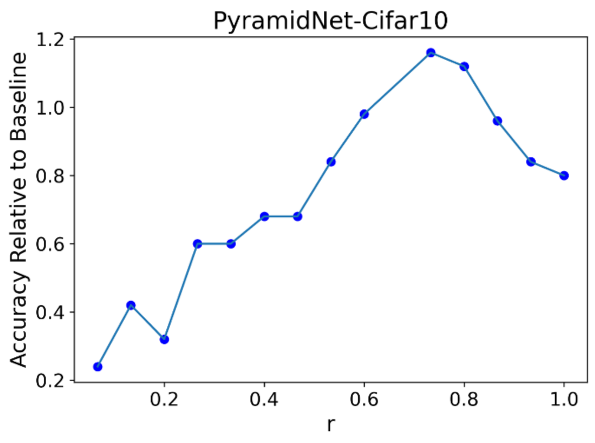

Figure 34 Unimodal Pattern Using RUA formulation

The same trend can be observed in the RandAugment [23] (Cubuk et al., 2020) paper too, reproduced here as Figure 35. This demonstrates that the unimodal relationship persists across different network and dataset sizes.

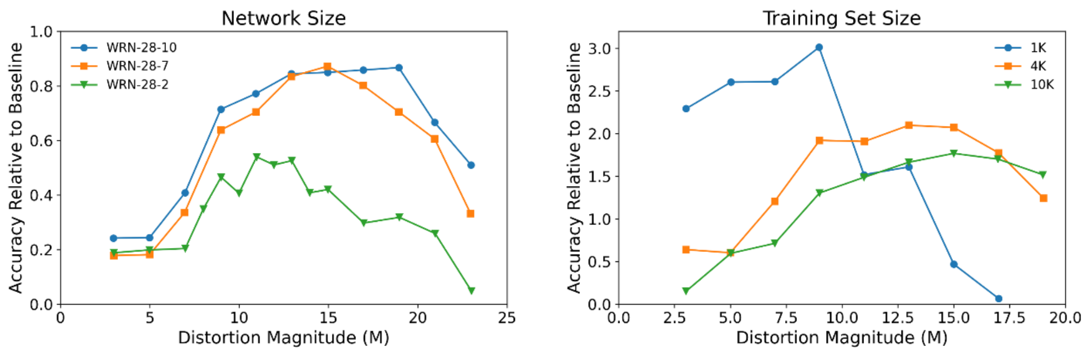

Figure 35 RandAugment Accuracy as Function of M for Various Network and Data Sizes

In light of this unimodal property, we can leverage algorithms that are more efficient than grid search to explore a larger search space using less computation. One such algorithm is golden-section search [48] (Kiefer, 1953). Golden-section search is a simple method that is widely used for finding the maximum or minimum of a unimodal function over a given interval. The pseudo code for golden-section search is given in Figure 36.

With golden-section search, every evaluation (after the first) of the search space will reduce the remaining search space by a constant factor of ~ *0.618* (inverse golden ratio). As a result, we can search over 90% of the domain of *r* using only 6 evaluations. This makes it practical to search over the entire training dataset, without having to resort to subsampling like Fast AA or PBA. Note that this search space reduction does not require any human expertise or intervention, allowing the method to be used as a truly automated manner.

```
Algorithm 1 Golden Section Search (Max-Finding)
Require: $f, a, b,$ maxIter
    $\phi_1, \phi_2 \leftarrow \frac{\sqrt{5}-1}{2}, \frac{3-\sqrt{5}}{2}$
    $h \leftarrow b - a$
    $c \leftarrow a + \phi_2 * h$
    $d \leftarrow a + \phi_1 * h$
    $y_c \leftarrow f(c)$
    $y_d \leftarrow f(d)$
    for i from 1 to maxIter do
        if $y_c > y_d$ then
            $b \leftarrow d$
            $d \leftarrow c$
            $y_d \leftarrow y_c$
            $h \leftarrow \phi_1 * h$
            $c \leftarrow a + \phi_2 * h$
            $y_c \leftarrow f(c)$
        else
            $a \leftarrow c$
            $c \leftarrow d$
            $y_c \leftarrow y_d$
            $h \leftarrow \phi_1 * h$
            $d \leftarrow a + \phi_1 * h$
            $y_d \leftarrow f(d)$
        end if
    end for
    if $y_c > y_d$ then
        return c
    else
        return d
    end if
```

Figure 36 Pseudo Code for Golden-Section Search

4.2.3 Search Space Design and Augmentation Parameters

After the search space reduction, we are left with one parameter *r*, which controls the global augmentation intensity. The exact manner of this control is given in Table 10 (right). A zero value of *r* means no augmentation, whereas a value of 1 achieves maximum augmentation.

Table 10 Augmentations and Their Associated Parameters

Augmentations	RandAugment(RA)	RUA
Identity	-	-
AutoContrast	-	-
Equalize	-	-
Rotate	$degree = \pm 30r$	$degree = U(-90r, 90r)$
Solarize*	$threshold = 256r$	$threshold = 256 - U(0, 256r)$
Posterize*	$shift = 8 - 4r$	$shift = U(0, 7r)$
Color*	$factor = 1.8r + 0.1$	$factr = 1 + U(-0.9r, 0.9r)$
Contrast*	$factor = 1.8r + 0.1$	$factr = 1 + U(-0.9r, 0.9r)$
Brightness*	$factor = 1.8r + 0.1$	$factr = 1 + U(-0.9r, 0.9r)$
Sharpness*	$factor = 1.8r + 0.1$	$factr = 1 + U(-0.9r, 0.9r)$
Shear-X	$coef = \pm 0.3r$	$coef = U(-0.5r, 0.5r)$
Shear-Y	$coef = \pm 0.3r$	$coef = U(-0.5r, 0.5r)$
Translate-X	$coef = \pm 100r$	$coef = U(-r, r) \times \frac{width}{3}$
Translate-Y	$coef = \pm 100r$	$coef = U(-r, r) \times \frac{height}{3}$

This is a conceptual divergence from RandAugment, as 6 of their 14 transformations are not set up to scale this way. These transformations are marked with a "*" in Table 10 (left). For example, the transformation intensity of *Solarize* and *Posterize* are inversely correlated with *r*. Moreover, *Color*, *Contrast*, *Brightness*, and *Sharpness* are all shifted in that they cause no augmentation when $r = 0.5$, whereas values closer to 0 or 1 lead to stronger alterations to the input.

In addition to aligning *r* with augmentation intensity, RUA also introduces non-deterministic parameter selection into augmentations. For example, rather than rotating an image exactly ± 30 degrees whenever the *Rotate* operation is applied, it would instead draw from a random uniform distribution to cover the augmentation space more thoroughly. The maximum intensity of certain augmentations are also increased to keep the expected intensity consistent in spite of the switch to uniform distributions. Each of these decisions will be justified with an ablation study in Section 4.3.2.

4.2.4 Selecting Maximum N

One question which must be answered when applying RUA is what value to use as N_{max}. While one may be content to use 10, since that was the extent of the RandAugment search space, other numbers may well be equally valid. To investigate further, a second grid search is performed using RUA augmentation parameters, and results indicate that large values of N_{max} may not be necessary in order to achieve a good performance (Figure 37), because the optimal zone appears as an off-diagonal patch covering wide range of N.

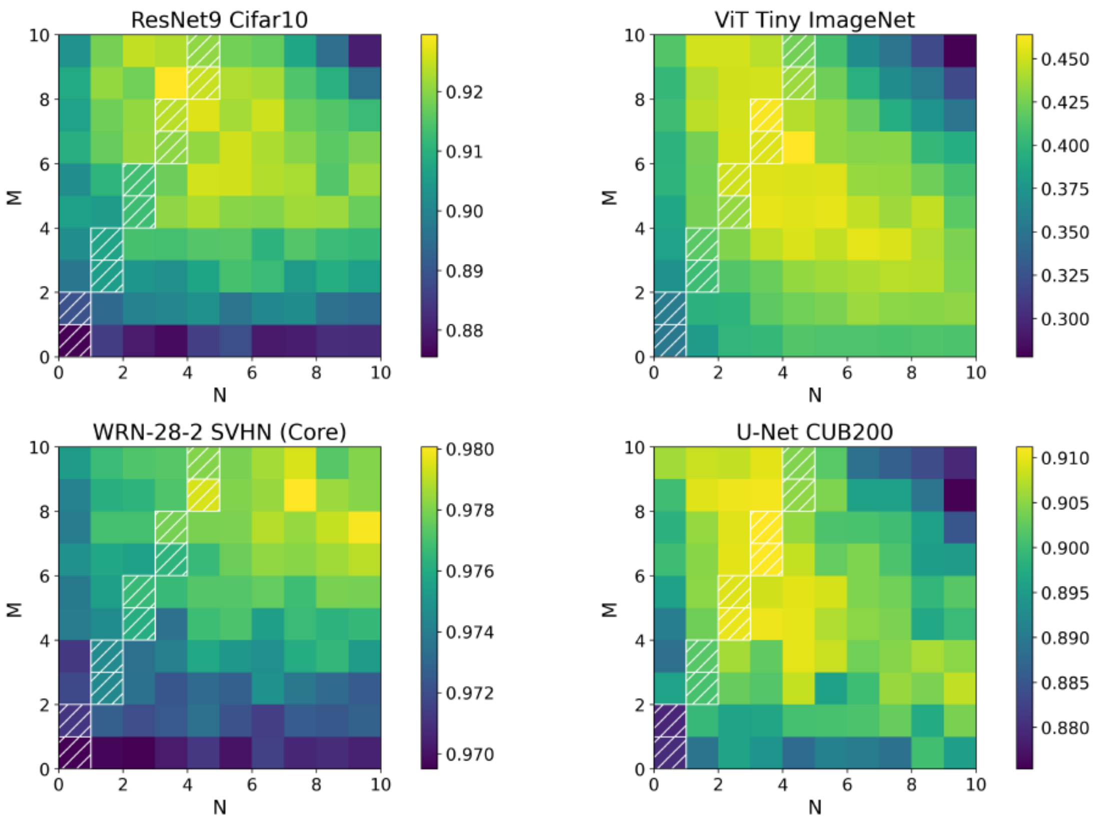

Figure 37 Test Performance as Function of M and N Using RUA Augmentation Parameters

This Phenomenon is a manifestation of a 2D Goldilocks principle. If there are two independent axes, each ranging from “not enough” up to “too much”, and each having similar magnitudes of effect on the end result, then the Goldilocks zone will be an off-diagonal patch connecting the two axes. Although the offset (y-intercept) of this patch will vary between problems, diagonally traversing the grid is guaranteed to pass through the zone eventually.

Based on this observation, the next thing to do is to examine what outcomes a user would achieve if they ran RUA using different values of N_{max} ranging from 1 to 10. This sensitivity analysis is shown in Figure 38. In these tests, setting $N_{max} > 5$ does not appear to provide any significant benefit, though small values like 1 or 2 can clearly be harmful, especially for ViT [49] (Deng et al., 2009) on Tiny ImageNet [50] (Li et al., 2016).

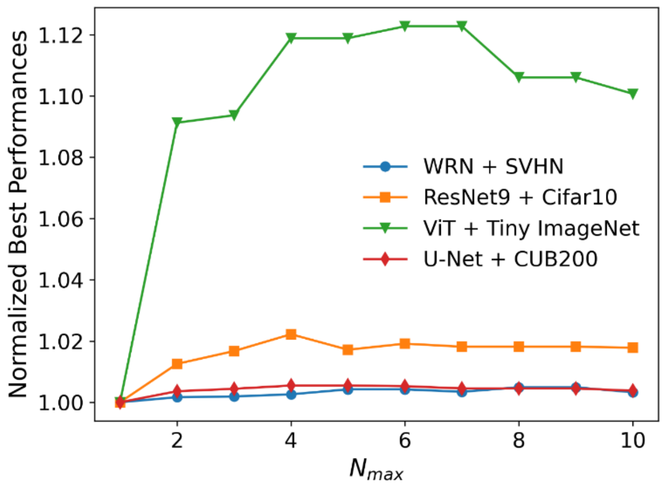

Figure 38 Best Performances Along Diagonal Search Path as Function of Nmax

Given that RUA is relatively insensitive to higher values of N_{max}, there are pragmatic reasons to choose values smaller than 10. Applying a large number of transformations during training can severely bottleneck the training speed. See Figure 39 for an example. For a widely-used hardware configuration (AWS EC2 P3.2xlarge), with any $N > 3$ the CPU-based preprocessing became rate limiting, especially once $N > 5$. This may be one reason why RandAugment never chose $N_{max} > 3$ in their sub-grid selections. With these factors in mind, N_{max} is selected as 5 for later experiments.

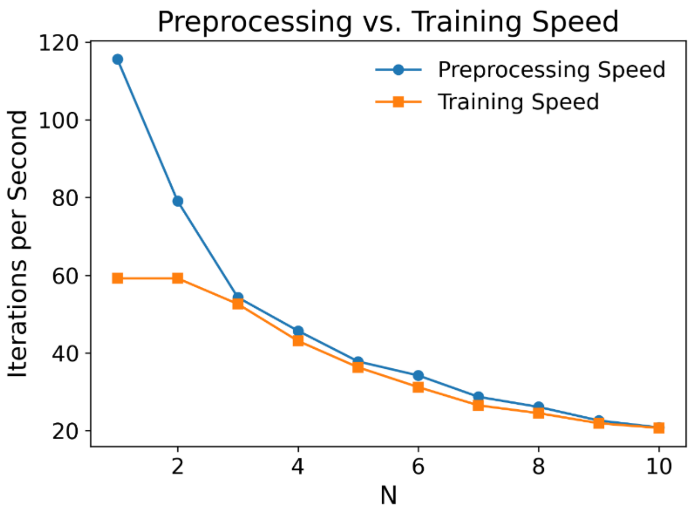

Figure 39 Preprocessing and Training Speed as Function of N. Measurements Were Taken on AWS Instance (8 core Intel Xeon CPU, NVIDIA Tesla V100 GPU)

4.3 Experiments

4.3.1 Performance Assessment

Design:

In order to perform a direct comparison with previous works, RUA is deployed in the same image classification training setting used by prior works on CIFAR10, CIFAR100, SVHN, and ImageNet. Details regarding the parameters used in each experiment are given in Table 11. There are a few things worth highlighting about experimental parameters.

To be consistent with previous works, default augmentations are also applied before and after applying RUA augmentation on different tasks. For example, pad-and-crop, horizontal flip, and cutout [51] (Devries and Taylor, 2017) are used on the CIFAR10 and CIFAR100 datasets.

For CIFAR10, RandAugment trained for 1800 epochs whereas the official implementation of PyramidNet [52] (Han et al., 2017) and ShakeDrop [53] (Yamada et al., 2019) trained for 300 epochs. This study picked 900 epochs as a compromise between different official implementation settings.

In every dataset 5k training samples will be held-out as evaluation data for selecting the best *r*. After selecting the best *r*, the hold-out set will be put back into the training set and train again with the optimal *r*. The test performance will be recorded at the end of that final training.

Table 11 Experiment Parameter Details

Dataset	CIFAR10	CIFAR10	CIFAR100	SVHN(Core)	ImageNet
Network	PyramidNet-272-200	WRN2810	WRN2810	WRN282	ResNet50
Epochs	900	200	200	200	180
Batch Size	128	128	128	128	4096
Preprocessing	Zscore	Zscore	Zscore	Divide by 255	None
Augmentations	[pad-and-crop, horizontal flip, RUA, cutout]	[pad-and-crop, horizontal flip, RUA, cutout]	[pad-and-crop, horizontal flip, RUA, cutout]	[RUA, cutout]	[random sized crop, horizontal flip, RUA]
Optimizer	SGD	SGD	SGD	SGD	SGD
Weight Decay	1e-4	5e-4	5e-4	5e-4	1e-4
Initial LR	0.1	0.1	0.1	0.1	1.6
LR Schedule	Cosine Decay	Cosine Decay	Cosine Decay	Cosine Decay	Multiply 0.1 at epochs 60, 120, 160
Momentum	0.9	0.9	0.9	0.9	0.9
Nmax	5	5	5	5	5
Best r	0.867	0.6	0.733	0.8	0.666

Results:

The final test results of RUA are shown in Table 12, where accuracy scores are from an average of 10 independent runs. The results of previous methods including the baseline, AA, Fast AA, PBA, and RA are taken from previous work [23] (Cubuk et al., 2020). The best candidate for each column is highlighted in bold.

Table 12 Experimental Results for RUA Compared With Previous Works

Methods	Search Space	Search Iterations	CIFAR10		CIFAR100	SVHN (Core)	ImageNet
			PyramidNet	WRN2810	WRN2810	WRN282	ResNet50
Baseline	-	-	97.3	96.1	81.2	96.7	76.3
AA	10^{32}	15000	**98.5**	**97.4**	82.9	98	77.6
Fast AA	10^{32}	200	98.3	97.3	82.7	-	77.6
PBA	10^{61}	16	**98.5**	**97.4**	83.3	-	-
RA	10^{2}	100	**98.5**	97.3	83.3	**98.3**	77.6
RUA(proposed)	**10**	**6**	**98.5**	**97.4**	**83.6**	98	**77.7**

The metrics used to evaluate each method are accuracy for each task, search space order, and the number of search iterations required. The accuracy measures the outcome of the search, with higher accuracy being more desirable. The search space order measures the complexity of the search space, the number of search iterations required measures the computation budget needed to perform the search. Each search iteration represents a training run on either full data or partial data.

Discussion:

As demonstrated in Table 12, RUA achieved equal or better test scores than previous state-of-the-art methods on 4 out of 5 tasks, while reducing the search space by an order of magnitude and cutting the training iteration requirements of the previous best method by more than 2 times.

For the CIFAR10 tasks, RUA is equivalent to the best prior methods, with one-tailed t-test p-values of 0.0017 and 0.034. For CIFAR100 and ImageNet our performance exceeds that of prior methods, with one-tailed t-test p-values of 0.002 and 0.039. On SVHN, despite being outperformed by RandAugment, RUA nonetheless achieved competitive performance on par with AutoAugment.

For search efficiency, the metric used is iteration required rather than GPU hours due to an unfortunate trend in recent publications where GPU hours are commonly compared using different hardware. For example, FastAA uses V100 GPUs to compare against AA which uses older P100 GPUs. Meanwhile, PBA measured their GPU hours on a Titan XP GPU instead. Since GPU speed will vary greatly across hardware generations, it is difficult to compare properly between papers without investing thousands of GPU hours re-running old methods. Perhaps RA noticed this problem since they did not attempt any type of speed comparison. On the other hand, by comparing the number of training runs required by each search method, it provides a useful metric by which future search algorithms can be benchmarked easily. Note that this metric is not a theoretical complexity, but the actual number of trainings required by each method. As a result of this metric, it can be clearly observed that RUA is faster than the previous fastest method by more than 2 times.

4.3.2 Ablation Study of Transformation Parameters

Design:

This section presents an ablation study on the various design decisions outlined in Section 4.2.3. There are in total of three design improvements proposed:

a) ***Aligned***: This indicates modification to the “*” transforms in RandAugment’s formula presented in Table 10.

b) ***Random***: This indicates the use of a random uniform distribution.

c) ***Expanded***: This indicates the use of expanded augmentation parameters.

Results:

The results of this study are given in Table 13. Each design improvement contains a binary option: True or False. All three design improvements can therefore form a truth table consisting of 8 rows. Row 1 is thus analogous to running RandAugment using the dimensionality reduction and golden section search routine, and row 8 is the full RUA method. The accuracy of each row’s configuration is the mean of 10 independent repeats.

Table 13 Ablation Study of RUA Design Decisions in Section 4.2.3

Number	Aligned	Random	Expanded	Accuracy
1	0	0	0	0.916
2	0	0	1	0.912
3	0	1	0	0.917
4	0	1	1	0.920
5	1	0	0	0.917
6	1	0	1	0.915
7	1	1	0	0.920
8	1	1	1	0.922

Discussion:

There are several takeaways that can be observed from Table 13. First, making the augmentations marked with “*” from Table 10 positively correlated with *r* is always beneficial. This can be seen through pairwise comparisons of rows 1 vs. 5, 2 vs. 6, 3 vs. 7, and 4 vs. 8.

The second takeaway is that using a random distribution to draw the transformation arguments is always beneficial. This can be seen through pairwise comparisons of rows 1 vs. 3, 2 vs. 4, 5 vs. 7, and 6 vs. 8.

Finally, increasing the maximum strength of augmentations (for example rotating ± 90 degrees rather than ± 30 degrees) is always deleterious on its own (rows 1 vs. 2 and 5 vs. 6), but advantageous when paired with random sampling (rows 3 vs. 4 and 7 vs. 8). This is not particularly surprising since larger effects under deterministic sampling will consistently and seriously distort an image, whereas under uniform sampling they permit a larger exploration of the distortion space while still more often sampling less extreme distortions. All told, the best design was to apply all three modifications (row 8).

4.4 Ethical Aspect of Automatic Data Augmentation

4.4.1 Fairness and Bias Issues with Automatic Augmentation

AutoML workflows, including automatic augmentation method, generally consist of two major components: a) optimization of a pre-defined objective w.r.t augmentation policy b) training of a machine learning model given the augmentation policy. The goal of different automatic augmentation methods [20, 21, 22, 23] is to provide better search efficiency; therefore, these methods only focus on the keyword “optimization” mentioned in component (a). However, what is equally important but usually overlooked is the pre-defined objective, because this objective directly relates to fairness and bias.

Fairness and bias issues will arise when the objective is not carefully selected. Commonly used objectives are performance metrics like accuracy, mean square error, dice coefficient, and so on. All these metrics share the same “averaging nature” inherited from their mathematical formula. While the averaging nature does offer an easy pathway in understanding group behavior, it inevitably causes fairness and bias issues when applying to human-related task. For example, optimizing accuracy on an imbalanced sample population leads to the model focusing more on the majority, because attending more to majority provides the most amount of accuracy gain. To avoid this, the objective must be carefully designed to ensure fairness and prevent biases.

Fortunately, the objective used by automatic augmentation methods does not require gradient computation, allowing any fairness related measures to be used as objective directly. In computer vision, explainable AI (XAI) methods like Grad-Cam [54] (Selvaraju et al., 2017) and Saliency Map [55] (Sundararajan et al., 2017) can help identify what regions of image that the network’s prediction is based on (Figure 40) Then in the objective function, we can add additional terms that penalize the learned model that relies on fairness-involved features like gender, race, etc.

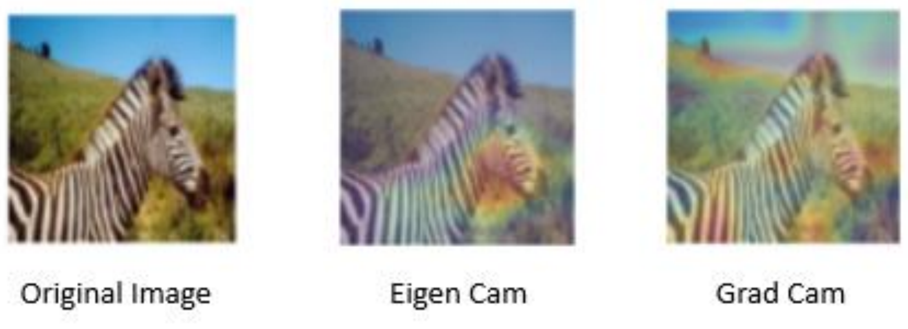

Figure 40 Explainable AI to Help Understand Network's Decision Making

Doing this will not only improve the fairness of the learned model, but also enhance model's robustness as an additional benefit. Because a model that achieves a good performance without exploiting obvious surface-level features indicates a better fundamental understanding of the learning task, and therefore more likely to be generalizable in the absence of those features.

4.4.2 Human's Role in Presence of Automatic Augmentation Approach

Automatic augmentation approach, just like any other automation system in the world, will gradually shift human's role from being lower-level participant to higher-level designer. To see human's role in the presence of automatic augmentation, it helps to view human's role in the presence of AutoML first.

Generally speaking, AutoML (automation of machine learning) has become a continuing trend with ever-increasing impact among machine learning community. It automates the low-level manual workflow that used to rely on human expertise with computation so that human can focus on higher-level designs instead. For example, in early days of deep learning, people used to spend considerable amount of time and efforts designing topology of a deep neural network and many well-recognized publications [5, 56, 57] were centered around this theme.

However, with the emergence of AutoML, many publications achieve state-of-the-art results either by directly leveraging an automation system [7, 8] or using the outcome of an automation system for further design [9, 10]. As a result of that, a substantial amount of research publications are shifting away from low-level workflow design towards higher-level system design. This gradual shift of research focus indicates a change of human’s role in the field of machine learning under automation.

Automatic augmentation, a sub-field of AutoML, is also contributing to the shifting of human’s role in its own way. Before automatic augmentation, given the under-augment and over-augment trade-off, researchers and machine learning practitioners used to hand-select augmentation policies then hand-tune its parameters. The manual process often results in both computation inefficiency due to the need of human intervention and underwhelming model performance due to lack of search space coverage.

With the introduction of automatic augmentation, the efforts of selecting and tuning of augmentation parameters will be taken care of by proven heuristics and efficient algorithms (like the algorithm used in this study). In addition, automatic augmentation can be used as a good abstraction layer to lower the cognitive complexity needed to perform augmentation, making the workflow more accessible by researchers and practitioners of different levels of expertise. Therefore, the hope is that in the presence of automatic augmentation, people will eventually either interact with the automatic system for better efficiency, or directly leverage the outcome of the search in a larger pipeline.

4.5 Chapter Conclusion

This chapter proposed Random Unidimensional Augmentation (RUA), an automated augmentation method providing several benefits over previous state-of-the-art methods. A summary of chapter highlights is presented below.

Prior works in auto-augmentation have tended to focus purely on presentations of methodology + results with little attention given to the design of the search space itself.

In contrast, this chapter use detailed visualizations and observations from real tasks (Section 4.2) to motivate the design of an auto-augmentation search space. Furthermore, this study is the first to successfully reduce the augmentation search space down to a single dimension, allowing for a simpler and more practical workflow.

This study can advance the knowledge of the auto-augmentation community through the discovery of a unimodal property in the augmentation parameter space, a property observable in the data of both this work and prior works, but never before recognized nor utilized.

RUA is the first to utilize this unimodal property, allowing it to be significantly more efficient than all previous state-of-the-art search methods while achieving better performance.

RUA's strength over prior state-of-the-art method is experimentally demonstrated on the same tasks across various network architectures and datasets. Unlike previous methods, RUA does not rely on any problem-specific human expertise, making the method truly automated and thus fit for use in conjunction with larger AutoML pipelines.

5 Conclusions

This dissertation is an effort of applying AutoML philosophy to deep learning from a perspective of data augmentation and optimization. It introduced deep learning with benefits and issues, then suggested AutoML as a potential solution to overcome these issues. Next, it proposed two novel automated workflows that could contribute to areas where AutoML is less recognized by the machine learning community.

The Automatic Rate Controller (ARC) is an AutoML system for optimization. It can monitor training signals in real-time and help deep learning models adjust learning rate towards better result with faster convergence speed. ARC successfully overcomes several limitations of conventional learning rate schedulers on training awareness, tuning efforts, transferability, and self-correction ability. In addition, ARC also outperforms several learning-based scheduling solutions in terms of performance, convergence speed, and computation overhead.

The Random Unidimensional Augmentation (RUA) is an AutoML workflow for data augmentation policy search. It can automatically search for optimal data augmentation policy and improve generalization performance for machine learning methods. Comparing with the prior best work in automated augmentation search, RUA reduces the search space by an order of magnitude while achieving equivalent or better results than existing state-of-the-art. Furthermore, RUA successfully eliminated the manual component in the previous best method, allowing it to be used a truly automated fashion with high efficiency.

As demonstrated by both methods, AutoML offers great efficiency benefits on various aspects of current deep learning workflows, and it has the potential to overcome the current deep learning challenges on efficiency improvement and cost reduction. However, fully automated deep learning still has a long way to go, as there are many other areas of deep learning that remain untapped to this day. For example, how to properly normalize the data from different data sources while preserving critical differences can be an AutoML problem. In optimization, how to efficiently search for an update rule and adapt the update rule dynamically during training process can also be an AutoML problem. Once all the missing pieces are found, how to combine everything automatically in an efficient workflow can be yet another AutoML problem.

The hope of the dissertation is to inspire more future works on AutoML for all aspects of deep learning. Once a practical fully automated deep learning workflow is created, the world would become much different. With further advancements in computing hardware, it may not be unrealistic to see a system that could automatically gather information, process and learn the information in fraction of a second. Once we make good use of that system, many dreams that only existed in science fiction movies might finally come true. For that reason, I wish it becoming a reality soon.

References

[1] Apple Machine Learning Research (2021). On-device Panoptic Segmentation for Camera Using Transformers. https://machinelearning.apple.com/research/panoptic-segmentation

[2] Paper with code. (2022, Feb 26). State of the art leaderboard. https://paperswithcode.com/sota

[3] SuperGLUE Benchmark. (2022, Feb 26). Leaderboard. https://super.gluebenchmark.com/leaderboard

[4] O.Ronneberger, P.Fischer, and T.Brox. U-net: Convolutional networks for biomedical image segmentation. In Medical Image Computing and Computer-Assisted Intervention(MICCAI), volume 9351 of LNCS, pages 234–241. Springer,2015.

[5] Kaiming He and Xiangyu Zhang and Shaoqing Ren and Jian Sun (2016). 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 (pp. 770–778). IEEE Computer Society.

[6] 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.

[7] 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, Stockholmsm ̈assan, Stockholm, Sweden, July 10-15, 2018, volume 80 of Proceedings of Machine Learning Research, pages 4092–4101. PMLR, 2018.

[8] 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.

[9] Mingxing Tan and Quoc V. Le (2019). EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. In Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA (pp. 6105–6114). PMLR.

[10] Pierre Foret and Ariel Kleiner and Hossein Mobahi and Behnam Neyshabur (2021). Sharpness-aware Minimization for Efficiently Improving Generalization. In 9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021. OpenReview.net.

[11] Schmidt, E.R.E., Zhao, H.T., Park, J.M. et al (2021). A human-specific modifier of cortical connectivity and circuit function. Nature 599, 640–644. https://doi.org/10.1038

[12] McCulloch WS, Pitts WH (1943). A logical calculus of the ideas immanent in nervous activity. Bull Math Biophys, 5:115–133.

[13] Wikipedia (2022). Neuron. https://en.wikipedia.org/wiki/Neuron

[14] Herculano-Houzel, S. (2012). The remarkable, yet not extraordinary, human brain as a scaled-up primate brain and its associated cost. Proceedings of the National Academy of Sciences, 109(Supplement 1), 10661–10668.

[15] M Abadi and Paul Barham and Jianmin Chen and Zhifeng Chen and Andy Davis and Jeffrey Dean and Matthieu Devin and Sanjay Ghemawat and Geoffrey Irving and Michael Isard and Manjunath Kudlur and Josh Levenberg and Rajat Monga and Sherry Moore and Derek Gordon Murray and Benoit Steiner and Paul A. Tucker and Vijay Vasudevan and Pete Warden and Martin Wicke and Yuan Yu and Xiaoqiang Zheng (2016). TensorFlow: A System for Large-Scale Machine Learning. In 12th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2016, Savannah, GA, USA, November 2-4, 2016 (pp. 265–283). USENIX Association.

[16] Priya Goyal, Piotr Doll ́ar, 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.

[17] 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.

[18] 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.

[19] Leslie N. Smith and Nicholay Topin. Super-convergence: Very fast training of residual networks using large learning rates. CoRR, abs/1708.07120, 2017.

[20] Ekin D. Cubuk, Barret Zoph, Dandelion Man ́e, Vijay Vasudevan, and Quoc V. Le. Autoaugment: Learning augmentation strategies from data. In IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019, pages 113–123. Computer Vision Foundation / IEEE, 2019.

[21] Sungbin Lim, Ildoo Kim, Taesup Kim, Chiheon Kim, and Sungwoong Kim. Fast autoaugment. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alch ́e-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, December 8-14, 2019, Vancouver, BC, Canada, pages 6662–6672, 2019.

[22] Daniel Ho, Eric Liang, Xi Chen, Ion Stoica, and Pieter Abbeel. Population based augmentation: Efficient learning of augmentation policy schedules. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA, volume 97 of Proceedings of Machine Learning Research, pages 2731–2741. PMLR, 2019.

[23] Ekin Dogus Cubuk, Barret Zoph, Jon Shlens, and Quoc Le. Randaugment: Practical automated data augmentation with a reduced search space. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual, 2020.

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

[25] 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.

[26] 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.

[27] Chien-Yao Wang, Hong-Yuan Mark Liao, Yueh-Hua Wu, Ping-Yang Chen, Jun-Wei Hsieh, & I-Hau Yeh (2020). CSPNet: A New Backbone that can Enhance Learning Capability of CNN. In 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR Workshops 2020, Seattle, WA, USA, June 14-19, 2020 (pp. 1571–1580). Computer Vision Foundation / IEEE.

[28] 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.

[29] 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.

[30] Maren Mahsereci and Philipp Hennig. Probabilistic line searches for stochastic optimization. The Journal of Machine Learning Research, 18(1):4262–4320, 2017.

[31] Sharan Vaswani, Aaron Mishkin, Issam Laradji, Mark Schmidt, Gauthier Gidel, and Simon Lacoste-Julien. Painless stochastic gradient: Interpolation, line-search, and convergence rates. In Advances in Neural Information Processing Systems, pages 3727–3740, 2019.

[32] 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.

[33] Christian Daniel, Jonathan Taylor, and Sebastian Nowozin. Learning step size controllers for robust neural network training. In Proceedings of the AAAI Conference on Artificial Intelligence, volume 30, 2016.

[34] Jun Shu, Yanwen Zhu, Qian Zhao, Deyu Meng, and Zongben Xu. Meta-lr-schedule-net: Learned lr schedules that scale and generalize, 2020.

[35] Zhen Xu, Andrew M. Dai, Jonas Kemp, and Luke Metz. Learning an adaptive learning rate schedule. CoRR,abs/1909.09712, 2019.

[36] Yuhuai Wu, Mengye Ren, Renjie Liao, and Roger Grosse. Understanding short-horizon bias in stochastic metaoptimization. arXiv preprint arXiv:1803.02021, 2018.

[37] 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.

[38] Sepp Hochreiter and J ürgen Schmidhuber. Long short-term memory. Neural computation, 9:1735–80, 12 1997.

[39] Ananya Kumar, Percy Liang, and Tengyu Ma. Verified uncertainty calibration. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alch´e-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, December 8-14, 2019, Vancouver, BC, Canada, pages 3787–3798, 2019.

[40] Davide Chicco and Giuseppe Jurman. The advantages of the matthews correlation coefficient (mcc) over f1 score and accuracy in binary classification evaluation. BMC genomics, 21(1):1–13, 2020.

[41] A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. Master's thesis, Department of Computer Science, University of Toronto, 2009.

[42] David C Page. Cifar10-fast dawn benchmark implementation. https://github.com/davidcpage/cifar10-fast. Accessed: 2022-02-27.

[43] Tsung-Yi Lin, Priya Goyal, Ross B. Girshick, Kaiming He, and Piotr Doll ́ar. 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.

[44] Mitchell P Marcus. Treebank-3 ldc99t42. web download. https://catalog.ldc.upenn.edu/LDC99T42. Accessed: 2022-02-27.

[45] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Fei-Fei Li. Imagenet: A large-scale hierarchical image database. In 2009 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2009), 20-25 June 2009, Miami, Florida, USA, pages 248–255. IEEE Computer Society, 2009.

[46] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. In Richard C. Wilson, Edwin R. Hancock, and William A. P. Smith, editors, Proceedings of the British Machine Vision Conference 2016, BMVC 2016, York, UK, September 19-22, 2016. BMVA Press, 2016.

[47] 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.

[48] J. Kiefer. Sequential minimax search for a maximum. Proceedings of the American Mathematical Society, 4(3):502–506, 1953.

[49] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. ICLR, 2021.

[50] Fei-Fei Li, Andrej Karpathy, and Justin Johnson. Tiny imagenet visual recognition challenge, 2016.

[51] Terrance Devries and Graham W. Taylor. Improved regularization of convolutional neural networks with cutout. CoRR, abs/1708.04552, 2017.

[52] Dongyoon Han, Jiwhan Kim, and Junmo Kim. Deep pyramidal residual networks. In 2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017, pages 6307–6315. IEEE Computer Society, 2017.

[53] Yoshihiro Yamada, Masakazu Iwamura, Takuya Akiba, and Koichi Kise. Shakedrop regularization for deep residual learning. IEEE Access, 7:186126–186136, 2019.

[54] Ramprasaath R. Selvaraju and Michael Cogswell and Abhishek Das and Ramakrishna Vedantam and Devi Parikh and Dhruv Batra (2017). Grad-CAM: Visual Explanations from Deep Networks via Gradient-Based Localization. In IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017 (pp. 618–626). IEEE Computer Society.

[55] Sundararajan, M., Taly, A. and Yan, Q., 2017. Axiomatic attribution for deep networks. Proceedings of the 34th International Conference on Machine Learning-Volume 70, pp. 3319--3328.

[56] Christian Szegedy and Vincent Vanhoucke and Sergey Ioffe and Jonathon Shlens and Zbigniew Wojna (2016). 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 (pp. 2818–2826). IEEE Computer Society.

[57] Gao Huang and Zhuang Liu and Kilian Q. Weinberger (2016). Densely Connected Convolutional Networks. CoRR, abs/1608.06993.

[58] FastEstimator Github. (2022). ARC. https://github.com/fastestimator/ARC

[59] FastEstimator Github. (2022). RUA. https://github.com/fastestimator/RUA

[60] Tsung-Yi Lin, Michael Maire, Serge J. Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, & C. Lawrence Zitnick (2014). Microsoft COCO: Common Objects in Context. In Computer Vision - ECCV 2014 - 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V (pp. 740–755). Springer.