BUDGETED TRAINING: RETHINKING DEEP NEURAL NETWORK TRAINING UNDER RESOURCE CONSTRAINTS

Anonymous authors
Paper under double-blind review

ABSTRACT

In most practical settings and theoretical analyses, one assumes that a model can be trained until convergence. However, the growing complexity of machine learning datasets and models may violate such assumptions. Indeed, current approaches for hyper-parameter tuning and neural architecture search tend to be limited by practical resource constraints. Therefore, we introduce a formal setting for studying training under the non-asymptotic, resource-constrained regime, i.e., budgeted training. We analyze the following problem: "given a dataset, algorithm, and *fixed* resource budget, what is the best achievable performance?" We focus on the number of optimization iterations as the representative resource. Under such a setting, we show that it is critical to adjust the learning rate schedule according to the given budget. Among budget-aware learning schedules, we find simple linear decay to be both robust and high-performing. We support our claim through extensive experiments with state-of-the-art models on ImageNet (image classification), Kinetics (video classification), MS COCO (object detection and instance segmentation), and Cityscapes (semantic segmentation). We also analyze our results and find that the key to a good schedule is budgeted convergence, a phenomenon whereby the gradient vanishes at the end of each allowed budget. We also revisit existing approaches for fast convergence and show that budget-aware learning schedules readily outperform such approaches under (the practical but under-explored) budgeted training setting.

1 INTRODUCTION

Deep neural networks have made an undeniable impact in advancing the state-of-the-art for many machine learning tasks. Improvements have been particularly transformative in computer vision (Huang et al., 2017b; He et al., 2017). Much of these performance improvements were enabled by an ever-increasing amount of labeled visual data (Russakovsky et al., 2015; Kuznetsova et al., 2018) and innovations in training architectures (Krizhevsky et al., 2012; He et al., 2016).

However, as training datasets continue to grow in size, we argue that an additional limiting factor is that of resource constraints for training. Conservative prognostications of dataset sizes – particularly for practical endeavors such as self-driving cars (Bojarski et al., 2016), assistive medical robots (Taylor et al., 2008), and medical analysis (Fatima & Pasha, 2017) – suggest one will train on datasets orders of magnitude larger than those that are publicly available today. Such planning efforts will become more and more crucial, because *in the limit, it might not even be practical to visit every training example before running out of resources* (Bottou, 1998; Rai et al., 2009).

We note that resource-constrained training already is *implicitly* widespread, as the vast majority of practitioners have access to limited compute. This is particularly true for those pursuing research directions that require a massive number of training runs, such as hyper-parameter tuning (Li et al., 2017) and neural architecture search (Zoph & Le, 2017; Cao et al., 2019; Liu et al., 2019).

Instead of asking "what is the best performance one can achieve given this data and algorithm?", which has been the primary focus in the field so far, we decorate this question with *budgeted training* constraints as follows: "what is the best performance one can achieve given this data and algorithm within the allowed budget?". Here, the *allowed budget* refers to a limitation on the total time,

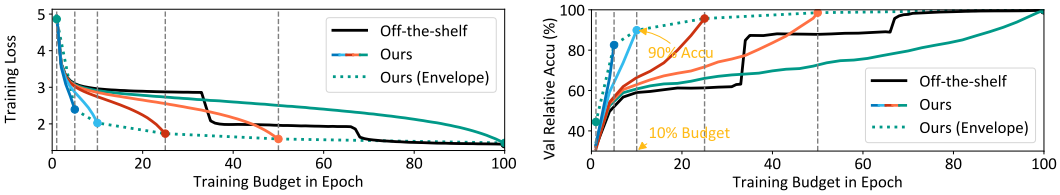


Figure 1: We formalize the problem of *budgeted training*, in which one maximizes performance subject to a fixed training budget. We find that a simple and effective solution is to adjust the learning rate schedule accordingly and anneal it to 0 at the end of the training budget. This significantly outperforms off-the-shelf schedules, particularly for small budgets. This plot shows several training schemes (solid curves) for ResNet-18 on ImageNet. The vertical axis in the right plot is normalized by the validation accuracy achieved by the full budget training. The dotted green curve indicates an efficient way of trading off computation with performance.

compute, or cost spent on training. More specifically, we focus on limiting the number of iterations. This allows us to abstract out the specific constraint without loss of generality since any one of the aforementioned constraints could be converted to a finite iteration limit. We make the underlying assumption that the network architecture is constant throughout training, though it may be interesting to entertain changes in architecture during training (Rusu et al., 2016; Wang et al., 2017).

Much of the theoretical analysis of optimization algorithms focuses on asymptotic convergence and optimality (Robbins & Monro, 1951; Nemirovski et al., 2009; Bottou et al., 2018), which implicitly makes use of an *infinite* compute budget. That said, there exists a wide body of work (Zinkevich, 2003; Kingma & Ba, 2015; Reddi et al., 2018; Luo et al., 2019) that provide performance bounds which depend on the iteration number, which apply even in the non-asymptotic regime. Our work differs in its exploration of maximizing performance for a fixed number of iterations. Importantly, the globally optimal solution may not even be achievable in our budgeted setting.

Given a limited budget, one obvious strategy might be data subsampling (Bachem et al., 2017; Sener & Savarese, 2018). However, we discover that a much more effective, simpler, and under-explored strategy is adopting budget-aware learning rate schedules — if we know that we are limited to a single epoch, one should tune the learning schedule accordingly. Such budget-aware schedules have been proposed in previous work (Feyzmahdavian et al., 2016; Lian et al., 2017), but often for a fixed learning rate that depends on dataset statistics. In this paper, we specifically point out *linearly* decaying the learning rate to 0 at the end of the budget, may be more robust than more complicated strategies suggested in prior work. Though we are motivated by budget-aware training, we find that a linear schedule is quite competitive for general learning settings as well. We verify our findings with state-of-the-art models on ImageNet (image classification), Kinetics (video classification), MS COCO (object detection and instance segmentation), and Cityscapes (semantic segmentation).

We conduct several diagnostic experiments that analyze learning rate decays under the budgeted setting. We first observe a statistical correlation between the learning rate and the *full* gradient magnitude (over the entire dataset). Decreasing the learning rate empirically results in a decrease in the full gradient magnitude. Eventually, as the former goes to zero, the latter vanishes as well, suggesting that the optimization has reached a critical point, if not a local minimum[1]. We call this phenomenon *budgeted convergence* and we find it generalizes across budgets. On one hand, it implies that one should decay the learning rate to zero at the end of the training, even given a small budget. On the other hand, it implies one should not aggressively decay the learning rate early in the optimization (such as the case with an exponential schedule) since this may slow down later progress. Finally, we show that linear budget-aware schedules outperform recently-proposed fast-converging methods that make use of adaptive learning rates and restarts.

Our main contributions are as follows:

- We introduce a formal setting for budgeted training based on training iterations and provide an alternative perspective for existing learning rate schedules.

[1]Whether such a solution is exactly a local minimum or not is debatable (see Sec 2).

- We discover that budget-aware schedules are handy solutions to budgeted training. Specifically, our proposed linear schedule is more simple, robust, and effective than prior approaches, for both budgeted and general training.
- We provide an empirical justification of the effectiveness of learning rate decay based on the correlation between the learning rate and the full gradient norm.

2 Related Work

Learning rates. Stochastic gradient descent dates back to Robbins & Monro (1951). The core is its update step: $w_t = w_{t-1} - \alpha_t g_t$, where t (from 1 to T) is the iteration, w are the parameters to be learned, g is the gradient estimator for the objective function[2] F, and α_t is the *learning rate*, also known as *step size*. Given base learning rate α_0, we can define the ratio $\beta_t = \alpha_t/\alpha_0$. Then the set of $\{\beta_t\}_{t=1}^T$ is called the *learning rate schedule*, which specifies how the learning rate should vary over the course of training. *Our definition differs slighter from prior art as it separates the base learning rate and learning rate schedule*. Learning rates are well studied for (strongly) convex cost surfaces and we include a brief review in Appendix H.

Learning rate schedule for deep learning. In deep learning, there is no consensus on the exact role of the learning rate. Most theoretical analysis makes the assumption of a small and constant learning rate (Du et al., 2018a;b; Hardt et al., 2016). For variable rates, one hypothesis is that large rates help move the optimization over large energy barriers while small rates help converge to a local minimum (Loshchilov & Hutter, 2016; Huang et al., 2017a; Kleinberg et al., 2018). Such hypothesis is questioned by recent analysis on mode connectivity, which has revealed that there does exist a descent path between solutions that were previously thought to be isolated local minima (Garipov et al., 2018; Dräxler et al., 2018; Gotmare et al., 2019). Despite a lack of theoretical explanation, the community has adopted a variety of heuristic schedules for practical purposes, two of which are particularly common:

- **step decay**: drop the learning rate by a multiplicative factor γ after every d epochs. The default for γ is 0.1, but d varies significantly across tasks.
- **exponential**: $\beta_t = \gamma^t$. There is no default parameter for γ and it requires manual tuning.

State-of-the-art codebases for standard vision benchmarks tend to employ step decay (Xie & Tu, 2015; Huang et al., 2017b; He et al., 2017; Carreira & Zisserman, 2017; Wang et al., 2018; Yin et al., 2019; Ma et al., 2019), whereas exponential decay has been successfully used to train Inception networks (Szegedy et al., 2015; 2016; 2017). In spite of their prevalence, these heuristics have not been well studied. Recent work proposes several new schedules (Loshchilov & Hutter, 2016; Smith, 2017; Hsueh et al., 2019), but much of this past work limits their evaluation to CIFAR and ImageNet. For example, SGDR (Loshchilov & Hutter, 2016) advocates for learning-rate restarts based on the results on CIFAR, however, we find the unexplained form of cosine decay in SGDR is more effective than the restart technique. Notably, Mishkin et al. (2017) demonstrate the effectiveness of linear rate decay with CaffeNet on downsized ImageNet. In our work, we rigorously evaluate on 5 standard vision benchmarks with state-of-the-art networks and under various budgets. Gotmare et al. (2019) also analyze learning rate restarts and in addition, the warm-up technique, but do not analyze the specific form of learning rate decay.

Adaptive learning rates. Adaptive learning rate methods (Tieleman & Hinton, 2012; Kingma & Ba, 2015; Reddi et al., 2018; Luo et al., 2019) adjust the learning rate according to the local statistics of the cost surface. Despite having better theoretical bounds under certain conditions, they do not generalize as well as momentum SGD for benchmark tasks that are much larger than CIFAR (Wilson et al., 2017). We offer new insights by evaluating them under the budgeted setting. We show fast descent can be trivially achieved through budget-aware schedules and aggressive early descent is not desirable for achieving good performance in the end.

[2]Note that g can be based on a single example, a mini-batch, the full training set, or the true data distribution. In most practical settings, momentum SGD is used, but we omit the momentum here for simplicity.

3 LEARNING RATES AND BUDGETS

3.1 BUDGET-AWARE SCHEDULES

Learning rate schedules are often defined assuming unlimited resources. As we argue, resource constraints are an undeniable practical aspect of learning. One simple approach for modifying an existing learning rate schedule to a budgeted setting is early-stopping. Fig 1 shows that one can dramatically improve results of early stopping by more than 60% by *tuning* the learning rate for the appropriate budget. To do so, we simply reparameterize the learning rate sequence with a quantity not only dependent on the absolute iteration t, but also the training budget T:

Definition (Budget-Aware Schedule). Let T be the training budget, t be the current step, then a training progress p is t/T. A *budget-aware learning rate schedule* is

$$\beta_p : p \mapsto f(p), \tag{1}$$

where $f(p)$ is the ratio of learning rate at step t to the base learning rate α_0.

At first glance, it might be counter-intuitive for a schedule to *not* depend on T. For example, for a task that is usually trained with 200 epochs, training 2 epochs will end up at a solution very distant from the global optimal no matter the schedule. In such cases, conventional wisdom from convex optimization suggests that one should employ a large learning rate (constant schedule) that efficiently descends towards the global optimal. However, in the non-convex case, we observe empirically that a better strategy is to systematically decay the learning rate in proportion to the total iteration budget.

Budge-Aware Conversion (BAC). Given a particular rate schedule $\beta_t = f(t)$, one simple method for making it budget-aware is to rescale it, *i.e.*, $\beta_p = f(pT_0)$, where T_0 is the budget used for the original schedule. For instance, a step decay for 90 epochs with two drops at 30 epoch and 60 epoch will convert to a schedule that drops at 1/3 and 2/3 training progress. Analogously, an exponential schedule 0.99^t for 200 epochs will be converted into $(0.99^{200})^p$.

It is worth noting that such an adaptation strategy already exists in well-known codebases (He et al., 2017) for training with limited schedules. Our experiments confirm the effectiveness of BAC as a general strategy for converting many standard schedules to be budget-aware (Tab 1). *For our remaining experiments, we regard BAC as a known technique and apply it to our baselines by default.*

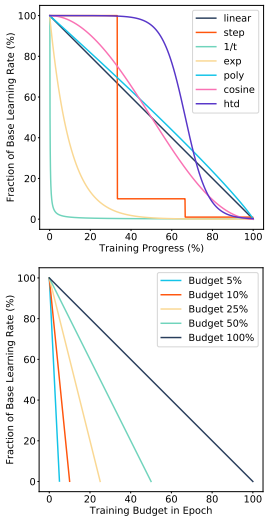
Budget	1%	5%	10%	25%	50%	100%
exp .99	.5848	.8030	.8352	.8888	.9072	.9320
BAC	**.6086**	**.8560**	**.8996**	**.9228**	**.9272**	N/A
step-d1	.5710	.8058	.8422	.8702	.8746	.9434
BAC	**.5880**	**.8662**	**.9066**	**.9312**	**.9392**	N/A

Table 1: Effectiveness of budget-aware conversion (BAC) on CIFAR-10 for image classification with ResNet-18 (He et al., 2016). The numbers are classification accuracy on the validation set. The 100% budget refers to training for 200 epochs. "step-d1" denotes step decay dropping once at training progress 50%. Please refer to Sec 4.1 for the complete setup.

Recent schedules: Interestingly, several recent learning rate schedules are implicitly defined as a function of progress $p = \frac{t}{T}$, and so are budget-aware by our definition:

- **poly** (Jia et al., 2014): $\beta_p = (1-p)^\gamma$. No parameter other than $\gamma = 0.9$ is used in published work.
- **cosine** (Loshchilov & Hutter, 2016): $\beta_p = \eta + \frac{1}{2}(1-\eta)(1+\cos(\pi p))$. η specify a lower bound for the learning rate, which defaults to zero.
- **htd** (Hsueh et al., 2019): $\beta_p = \eta + \frac{1}{2}(1-\eta)(1-\tanh(L+(U-L)p))$. Here η has the same representation as in cosine. It is reported that $L = -6$ and $U = 3$ performs the best.

The poly schedule is a feature in Caffe (Jia et al., 2014) and adopted by the semantic segmentation community (Chen et al., 2018; Zhao et al., 2017). The cosine schedule is a byproduct in work that promotes learning rate restarts (Loshchilov & Hutter, 2016). The htd schedule is recently proposed (Hsueh et al., 2019), which however, contains only limited empirical evaluation. None of these analyze their budget-aware property or provides intuition for such forms of decay. These schedules were



Budget	1%	5%	10%	25%	50%	100%
const	.5830	.7968	.8410	.8662	.8726	.8790
exp .95	.4796	.7554	.8574	.9140	.9294	.9458
exp .97	.5538	.8356	.8940	.9308	.9420	.9552
exp .99	.6086	.8560	.8996	.9228	.9272	.9320
step-d1	.5806	.8648	.9066	.9318	.9408	.9434
step-d2	.5544	.8328	.9042	.9338	.9464	.9534
step-d3	.4882	.7942	.8872	.9260	.9436	.9532
htd	.6430	.8878	.9224	.9434	.9510	.9552
cosine	.6308	.8856	.9222	.9444	.9530	.9554
poly	.6584	.8912	.9244	.9416	.9494	.9534
linear	.6654	.8920	.9218	.9412	.9546	.9562

Figure 2: We normalize various learning rate schedules by training progress (top-left). Our solution to budgeted training is simple and universal — we decrease the learning rate linearly across the entire given budget (bottom-left). We compare these learning rate schedules on CIFAR-10 (right). The 1st, 2nd and the 3rd place under each budget are color coded. The number here is the classification accuracy and each one is the median of 3 independent runs. "step-dx" denotes decay x times at even intervals with $\gamma = 0.1$. For "exp" and "step" schedules, BAC (Sec 3.1) is applied in place of early stopping. We can see linear schedule surpasses other schedules under almost all budgets.

treated as "yet another schedule". However, our definition of budget-aware makes these schedules stand out as a general family.

3.2 Linear Schedule

Inspired by existing budget-aware schedules, we borrow an even simpler schedule from the simulated annealing literature (Kirkpatrick et al., 1983; McAllester et al., 1997; Nourani & Andresen, 1998)[3]:

$$\textbf{linear} : \beta_p = 1 - p. \tag{2}$$

In Fig 2 (left), we compare linear schedule with various existing schedules under the budget-aware setting. Note that this linear schedule is completely parameter-free. This property is particularly desirable in budgeted training, where little budget exists for tuning such a parameter. The excellent generalization of linear schedule across budgets (shown in the next section) might imply that the cost surface of deep learning is to some degree self-similar. Note that a linear schedule, together with other recent budget-aware schedules, produces a constant learning rate in the asymptotic limit *i.e.*, $\lim_{T\to\infty}(1 - t/T) = 1$. Consequently, such practically high-performing schedules tend to be ignored in theoretical convergence analysis (Robbins & Monro, 1951; Bottou et al., 2018).

4 Experiments

In this section, we first compare linear schedule against other existing schedules on the small CIFAR-10 dataset and then on a broad suite of vision benchmarks. The CIFAR-10 experiment is designed to extensively evaluate each learning schedule while the vision benchmarks are used to verify the observation on CIFAR-10. We provide important implementation settings in the main text while

[3] A link between SGD and simulated annealing has been recognized decades ago, where learning rate plays the role of temperature control (Bottou, 1991). Therefore, cooling schedules in simulated annealing can be transferred into learning rate schedules for SGD.

Budget	1%	5%	10%	25%	50%	100%	1%	5%	10%	25%	50%	100%
Image classification on ImageNet with ResNet							Object detection on COCO with Mask-RCNN					
step	.2033	.5205	.5959	.6558	.6798	**.6939**	.0493	.2005	.2539	.3144	.3528	.3763
linear	**.3080**	**.5721**	**.6231**	**.6632**	**.6812**	.6932	**.0514**	**.2092**	**.2627**	**.3215**	**.3574**	**.3794**
Video classification on Kinetics with I3D							Instance segmentation on COCO with Mask-RCNN					
step	.2952	.4966	.5681	.6449	.6858	.7124	.0487	.1926	.2392	.2906	.3198	**.3399**
linear	**.3296**	**.5304**	**.5964**	**.6638**	**.6998**	**.7215**	**.0511**	**.1988**	**.2460**	**.2943**	**.3242**	.3395

Table 2: Robustness of linear schedule across budgets, tasks and architectures. Linear schedule significantly outperforms step decay given limited budgets. Note that the off-the-shelf decay for each dataset has different parameters optimized for the specific dataset. For all step decay schedules, BAC (Sec 3.1) is applied to boost their budgeted performance. To reduce stochastic noise, we report the median of 3 independent runs for all the numbers. See Sec 4.2 for the metrics of each task (the higher the better for all tasks).

leaving the rest of the details to Appendix L. In addition, we provide in Appendix A the evaluation with a large number of random architectures in the setting of neural architecture search.

4.1 CIFAR

CIFAR-10 (Krizhevsky & Hinton, 2009) is a dataset that contains 70,000 tiny images (32×32). Given its small size, it is widely used for validating novel architectures. We follow the standard setup for dataset split (Huang et al., 2017b), which is randomly holding out 5,000 from the 50,000 training images to form the validation set. For each budget, we report the best validation accuracy among epochs up till the end of the budget. We use ResNet-18 (He et al., 2016) as the backbone architecture and utilize SGD with base learning rate 0.1, momentum 0.9, weight decay 0.0005 and a batch size 128.

We study learning schedules in several groups: (a) constant (equivalent to not using any schedule). (b) & (c) exponential and step decay, both of which are commonly adopted schedules. (d) htd (Hsueh et al., 2019), a quite recent addition and not yet adopted in practice . We take the parameters with the best-reported performance $(-6, 3)$. Note that this schedule decays much slower initially than the linear schedule (Fig 2). (e) the smooth-decaying schedules (small curvature), which consists of cosine (Loshchilov & Hutter, 2016), poly (Jia et al., 2014) and the linear schedule.

As shown in Fig 2 (right), the group of schedules that are budget-aware by our definition, outperform other schedules under all budgets. The linear schedule in particular, performs best most of the time including the typical full budget case. Noticeably, when exponential schedule is well-tuned for this task ($\gamma = 0.97$), it fails to generalize across budgets. In comparison, the budget-aware group does not require tuning but generalizes much better.

Within the budget-aware schedules, cosine, poly and linear achieve very similar results. This is expected due to the fact that their numerical similarity at each step (Fig 2). These results might indicate that *the key for a robust budgeted-schedule is to decay smoothly to zero.* Based on these observations and results, we suggest linear schedule should be the "go-to" budget-aware schedule.

4.2 Vision Benchmarks

In the previous section, we showed that linear schedule achieves excellent performance on CIFAR-10, in a relatively toy setting. In this section, we study the comparison and its generalization to practical large scale datasets with various state-of-the-art architectures. In particular, we set up experiments to validate the performance of linear schedule across tasks and budgets.

Ideally, one would like to see the performance of all schedules in Fig 2 on vision benchmarks. Due to resource constraints, we include only the off-the-shelf step decay and the linear schedule. Note our CIFAR-10 experiment suggests that using cosine and poly will achieve similar performance as linear, which are already budget-aware schedules given our definition, so we focus on linear schedule in this section. More evaluation between cosine, poly and linear can be found in Appendix A & E.

We consider the following suite of benchmarks spanning many flagship vision challenges:

Image classification on ImageNet. ImageNet (Russakovsky et al., 2015) is a widely adopted standard for image classification task. We use ResNet-18 (He et al., 2016) and report the top-1 accuracy on the validation set with the best epoch. We follow the step decay schedule used in (Huang et al., 2017b; PyTorch, 2019), which drops twice at uniform interval ($\gamma = 0.1$ at $p \in \{\frac{1}{3}, \frac{2}{3}\}$). We set the full budget to 100 epochs (10 epochs longer than typical) for easier computation of the budget.

Video classification on Kinetics with I3D. Kinetics (Kay et al., 2017) is a large-scale dataset of YouTube videos focusing on human actions. We use the 400-category version of the dataset and a variant of I3D (Carreira & Zisserman, 2017) with training and data processing code publicly available (Wang et al., 2018). Top-1 accuracy of the final model is used for evaluating the performance. We follow the 4-GPU 300k iteration schedule (Wang et al., 2018), which features a step decay that drops 2 times with $\gamma = 0.1$ at $p \in \{\frac{1}{2}, \frac{5}{6}\}$.

Object detection and instance segmentation on MS COCO. MS COCO (Lin et al., 2014) is a widely recognized benchmark for object detection and instance segmentation. We use the standard COCO AP (averaged over IoU thresholds) metric for evaluating bounding box output and instance mask output. The AP of the final model on the validation set is reported in our experiment. We use the challenge winner Mask R-CNN (He et al., 2017) with a ResNet-50 backbone and follows its setup. For training, we adopt the 1x schedule (90k iterations), and the off-the-shelf (He et al., 2017) step decay that drops 2 times with $\gamma = 0.1$ at $p \in \{\frac{2}{3}, \frac{8}{9}\}$.

In addition, we include results on Cityscapes (Cordts et al., 2016) for semantic segmentation in Appendix E. Note that all off-the-shelf methods for these vision benchmarks employ SGD with momentum 0.9 and we adopt the same setting in our experiments.

If we factor in the dimension of budgets, Tab 2 shows a clear advantage of linear schedule over step decay. For example, on ImageNet, linear achieves 51.5% improvement at 1% of the budget. Next, we consider the full budget setting, where we simply swap out the off-the-shelf schedule with linear schedule. We observe better (video classification) or comparable (other tasks) performance after the swap. This is surprising given the fact that linear schedule is parameter-free and thus not optimized for the particular task or network.

In summary, *the smoothly decaying linear schedule is a simple and effective strategy for budgeted training*. It significantly outperforms traditional step decay given limited budgets, while achieving comparable performance with the normal full budget setting.

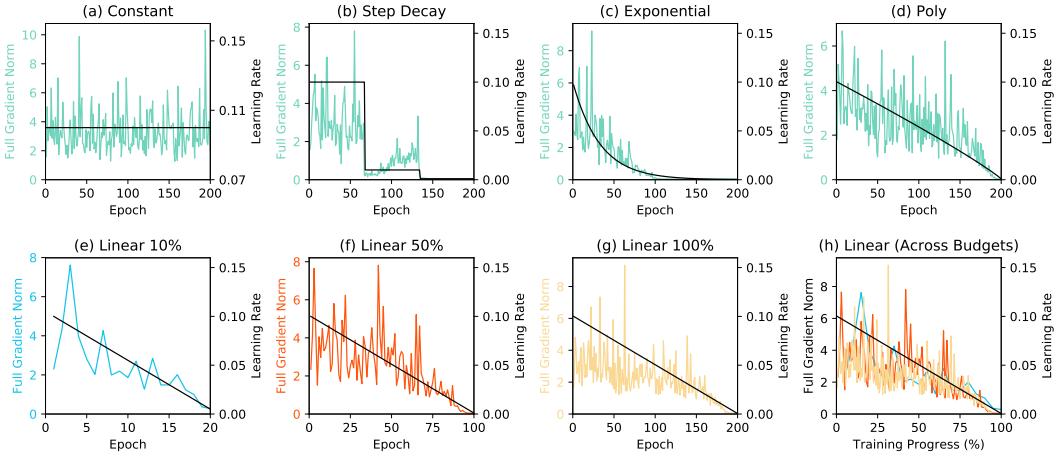


Figure 3: Budgeted convergence: full gradient norm $||g_t^*||$ vanishes over time (color curves) as learning rate α_t (black curves) decays. The first row shows that the dynamics of full gradient norm correlate with the corresponding learning rate schedule while the second row shows that such phenomena generalize across budgets for budget-aware schedules. Such generalization is most obvious in plot (h), which overlays the full gradient norm across different budgets. If a schedule does not decay to 0, the gradient norm does not vanish. For example, if we train a budget-unaware exponential schedule for 50 epochs (c), the full gradient norm at that time is around 1.5, suggesting this is a schedule with insufficient final decay of learning rate.

Schedule	Best Progress	Schedule	Best Progress
const	81.2% $\pm$ 16.1%	step-d2	90.5% $\pm$ 9.0%
linear	98.6% $\pm$ 1.6%	poly	99.1% $\pm$ 1.3%

Table 3: Where does one expect to find the model with the highest validation accuracy within the training progress? Here we show the best checkpoint location measured in training progress p and averaged for each schedule across budgets greater or equal than 10% and 3 different runs.

5 Discussion

In this section, we summarize our empirical analysis with a *desiderata* of properties for effective budget-aware learning schedules. We highlight those are inconsistent with conventional wisdom and follow the experimental setup in Sec 4.1 unless otherwise stated.

Desideratum: budgeted convergence. Convergence of SGD under non-convex objectives is measured by $\lim_{t\to\infty} \mathbb{E}[||\nabla F||^2] = 0$ (Bottou et al., 2018). Intuitively, one should terminate the optimization when no further local improvement can be made. What is the natural counterpart for "convergence" within a budget? For a dataset of N examples $\{(x_i, y_i)\}_{i=1}^N$, let us write the full gradient as $g_t^* = \frac{1}{N}\sum_{i=1}^N \nabla F(x_i, y_i)$. We empirically find that the dynamics of $||g_t^*||$ over time highly correlates with the learning rate α_t (Fig 3). As the learning rate vanishes for budget-aware schedules, so does the gradient magnitude. We call this "vanishing gradient" phenomenon *budgeted convergence*. This correlation suggests that decaying schedules to near-zero rates (and using BAC) may be more effective than early stopping. As a side note, budgeted convergence resonates with classic literature that argues that SGD behaves similar to simulated annealing (Bottou, 1991). Given that α_t and $||g_t^*||$ decrease, the overall update $||-\alpha_t g_t||$ also decreases[4]. In other words, large moves are more likely given large learning rates in the beginning, while small moves are more likely given small learning rates in the end. However, the exact mechanism by which the learning rate influences the gradient magnitude remains unclear.

Desideratum: don't waste the budget. Common machine learning practise often produces multiple checkpointed models during a training run, where a validation set is used to select the best one. Such additional optimization is wasteful in our budgeted setting. Tab 3 summarizes the progress point at which the best model tends to be found. Step decay produces an optimal model somewhat towards the end of the training, while linear and poly are almost always optimal at the precise end of the training. This is especially helpful for state-of-the-art models where evaluation can be expensive. For example, validation for Kinetics video classification takes several hours. Budget-aware schedules require validation on only the last few epochs, saving additional compute.

[4]Note that the momentum in SGD is used, but we assume vanilla SGD to simplify the discussion, without losing generality.

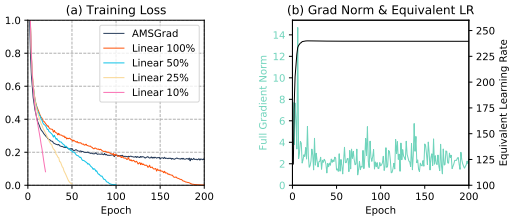


Figure 4: Comparing AMSGrad (Reddi et al., 2018) with linear schedule. (a) while AMSGrad makes fast initial descent of the training loss, it is surpassed at each given budget by the linear schedule. (b) budgeted convergence is not observed for AMSGrad — the full gradient norm $||g_t^*||$ does not vanish (color curves). Comparing to a momentum SGD, AMSGrad recommends magnitudes larger learning rate $\widetilde{\beta}_t$ (black curve).

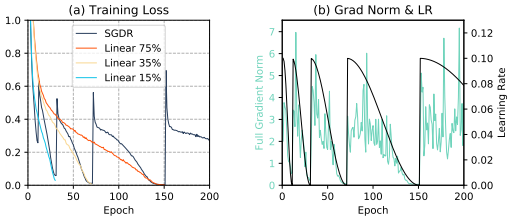


Figure 5: Comparing SGDR (Loshchilov & Hutter, 2016) with linear schedules. (a) SGDR makes slightly faster initial descent of the training loss, but is surpassed at each given budget by the linear schedule. (b) for SGDR, the correlation between full gradient norm $||g_t^*||$ and learning rate α_t is also observed. Warm restart does not help to achieve better budgeted performance.

Aggressive early descent. Guided by asymptotic convergence analysis, faster descent of the objective might be an apparent desideratum of an optimizer. Many prior optimization methods explicitly call for faster decrease of the objective (Kingma & Ba, 2015; Clevert et al., 2016; Reddi et al., 2018). In contrast, we find that one should not employ aggressive early descent because large learning rates can prevent budgeted convergence. Consider AMSGrad (Reddi et al., 2018), an adaptive learning rate that addresses a convergence issue with the widely-used Adam optimizer (Kingma & Ba, 2015). Fig 4 (a) shows that while AMSGrad does quickly descend over the training objective, it still underperforms budget-aware linear schedules over *any* given training budget. To examine why, we derive the equivalent rate $\widetilde{\beta}_t$ for AMSGrad (Appendix B) and show that it is dramatically larger than our defaults, suggesting the optimizer is too aggressive. We include more adaptive methods for evaluation in Appendix D.

Warm restarts. SGDR (Loshchilov & Hutter, 2016) explores periodic schedules, in which each period is a cosine scaling. The schedule is intended to escape local minima, but its effectiveness has been questioned (Gotmare et al., 2019). Fig 5 shows that SDGR has faster descent but is inferior to budget-aware schedules for *any* budget (similar to the adaptive optimizers above). Additional comparisons can be found in Appendix F. Whether there exists a method that achieves promising anytime performance and budgeted performance at the same time remains an open question.

6 Conclusion

This paper introduces a formal setting for budgeted training. Under this setup, we observe that a simple linear schedule, or any other smooth-decaying schedules can achieve much better performance. Moreover, the linear schedule even offers comparable performance on existing visual recognition tasks for the typical full budget case. In addition, we analyze the intriguing properties of learning rate schedules under budgeted training. We find that the learning rate schedule controls the gradient magnitude regardless of training stage. This further suggests that SGD behaves like simulated annealing and the purpose of a learning rate schedule is to control the stage of optimization.

References

Olivier Bachem, Mario Lucic, and Andreas Krause. Practical coreset constructions for machine learning. *arXiv preprint arXiv:1703.06476*, 2017.

Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D. Jackel, Miguel Pozuelo Monfort, Urs Muller, Jiakai Zhang, Xin Zhang, Junbo Jake Zhao, and Karol Zieba. End to end learning for self-driving cars. *CoRR*, abs/1604.07316, 2016.

Léon Bottou. Stochastic gradient learning in neural networks. In *Proceedings of Neuro-Nîmes 91*, Nimes, France, 1991. EC2. URL `http://leon.bottou.org/papers/bottou-91c`.

Léon Bottou. Online algorithms and stochastic approximations. In David Saad (ed.), *Online Learning and Neural Networks*. Cambridge University Press, Cambridge, UK, 1998. URL `http://leon.bottou.org/papers/bottou-98x`. revised, oct 2012.

Léon Bottou, Frank E. Curtis, and Jorge Nocedal. Optimization methods for large-scale machine learning. *SIAM Review*, 60:223–311, 2018.

Han Cai, Tianyao Chen, Weinan Zhang, Yong Yu, and Jun Wang. Efficient architecture search by network transformation. In *AAAI*, 2018.

Shengcao Cao, Xiaofang Wang, and Kris M Kitani. Learnable embedding space for efficient neural architecture compression. *ICLR*, 2019.

João Carreira and Andrew Zisserman. Quo vadis, action recognition? a new model and the kinetics dataset. *CVPR*, pp. 4724–4733, 2017.

Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L. Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *TPAMI*, 40:834–848, 2018.

Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *ICLR*, 2016.

Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. In *CVPR*, 2016.

Felix Dräxler, Kambis Veschgini, Manfred Salmhofer, and Fred A. Hamprecht. Essentially no barriers in neural network energy landscape. In *ICML*, 2018.

Simon S Du, Jason D Lee, Haochuan Li, Liwei Wang, and Xiyu Zhai. Gradient descent finds global minima of deep neural networks. *arXiv preprint arXiv:1811.03804*, 2018a.

Simon S. Du, Jason D. Lee, Yuandong Tian, Barnabás Póczos, and Aarti Singh. Gradient descent learns one-hidden-layer cnn: Don't be afraid of spurious local minima. In *ICML*, 2018b.

Meherwar Fatima and Maruf Pasha. Survey of machine learning algorithms for disease diagnostic. *Journal of Intelligent Learning Systems and Applications*, 9(01):1, 2017.

Hamid Reza Feyzmahdavian, Arda Aytekin, and Mikael Johansson. An asynchronous mini-batch algorithm for regularized stochastic optimization. *IEEE Transactions on Automatic Control*, 61 (12):3740–3754, 2016.

Timur Garipov, Pavel Izmailov, Dmitrii Podoprikhin, Dmitry P. Vetrov, and Andrew Gordon Wilson. Loss surfaces, mode connectivity, and fast ensembling of dnns. In *NeurIPS*, 2018.

Akhilesh Gotmare, Nitish Shirish Keskar, Caiming Xiong, and Richard Socher. A closer look at deep learning heuristics: Learning rate restarts, warmup and distillation. *ICLR*, 2019.

Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017.

Moritz Hardt, Benjamin Recht, and Yoram Singer. Train faster, generalize better: Stability of stochastic gradient descent. In *ICML*, 2016.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CVPR*, pp. 770–778, 2016.

Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross B. Girshick. Mask r-cnn. *ICCV*, pp. 2980–2988, 2017.

Bo Yang Hsueh, Wei Li, and I-Chen Wu. Stochastic gradient descent with hyperbolic-tangent decay on classification. *WACV*, 2019.

Gao Huang, Yixuan Li, Geoff Pleiss, Zhuang Liu, John E. Hopcroft, and Kilian Q. Weinberger. Snapshot ensembles: Train 1, get m for free. *ICLR*, 2017a.

Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. Densely connected convolutional networks. *CVPR*, pp. 2261–2269, 2017b.

Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. In *ACM Multimedia*, pp. 675–678, 2014.

Will Kay, João Carreira, Karen Simonyan, Brian Zhang, Chloe Hillier, Sudheendra Vijayanarasimhan, Fabio Viola, Tim Green, Trevor Back, Apostol Natsev, Mustafa Suleyman, and Andrew Zisserman. The kinetics human action video dataset. *CoRR*, abs/1705.06950, 2017.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *ICLR*, 2015.

Scott Kirkpatrick, C Daniel Gelatt, and Mario P Vecchi. Optimization by simulated annealing. *science*, 220(4598):671–680, 1983.

Robert D. Kleinberg, Yuanzhi Li, and Yang Yuan. An alternative view: When does sgd escape local minima? In *ICML*, 2018.

Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. *NIPS*, 60:84–90, 2012.

Alina Kuznetsova, Hassan Rom, Neil Alldrin, Jasper Uijlings, Ivan Krasin, Jordi Pont-Tuset, Shahab Kamali, Stefan Popov, Matteo Malloci, Tom Duerig, and Vittorio Ferrari. The open images dataset v4: Unified image classification, object detection, and visual relationship detection at scale. *arXiv:1811.00982*, 2018.

Lisha Li, Kevin G. Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet S. Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *JMLR*, 18:185:1–185:52, 2017.

Xiangru Lian, Ce Zhang, Huan Zhang, Cho-Jui Hsieh, Wei Zhang, and Ji Liu. Can decentralized algorithms outperform centralized algorithms? a case study for decentralized parallel stochastic gradient descent. In *NIPS*, pp. 5330–5340, 2017.

Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *ECCV*, pp. 740–755, 2014.

Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. *ICLR*, 2019.

Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. 2016.

Liangchen Luo, Yuanhao Xiong, Yan Liu, and Xu Sun. Adaptive gradient methods with dynamic bound of learning rate. *ICLR*, 2019.

W. Ma, S. Wang, R. Hu, Y. Xiong, and R. Urtasun. Deep rigid instance scene flow. In *CVPR*, pp. 1–9, 2019.

David McAllester, Bart Selman, and Henry Kautz. Evidence for invariants in local search. In *AAAI*, pp. 321–326, 1997.

Dmytro Mishkin, Nikolay Sergievskiy, and Jiri Matas. Systematic evaluation of convolution neural network advances on the imagenet. *Computer Vision and Image Understanding*, 161:11–19, 2017.

Arkadi Nemirovski, Anatoli Juditsky, Guanghui Lan, and Alexander Shapiro. Robust stochastic approximation approach to stochastic programming. *SIAM Journal on Optimization*, 19:1574–1609, 2009.

Yaghout Nourani and Bjarne Andresen. A comparison of simulated annealing cooling strategies. *Journal of Physics A: Mathematical and General*, 31(41):8373, 1998.

Hieu Pham, Melody Y Guan, Barret Zoph, Quoc V Le, and Jeff Dean. Efficient neural architecture search via parameter sharing. *ICML*, 2018.

PyTorch. ImageNet training in PyTorch v1.0.1, 2019. Retrieved from `https://github.com/pytorch/examples/tree/master/imagenet` on Mar 4, 2019.

Piyush Rai, Hal Daumé, and Suresh Venkatasubramanian. Streamed learning: one-pass svms. In *IJCAI*, 2009.

Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. *AAAI*, 2019.

Sashank J Reddi, Satyen Kale, and Sanjiv Kumar. On the convergence of adam and beyond. In *ICLR*, 2018.

Herbert Robbins and Sutton Monro. A stochastic approximation method. *The annals of mathematical statistics*, pp. 400–407, 1951.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *IJCV*, 115(3):211–252, 2015.

Andrei A Rusu, Neil C Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks. *arXiv preprint arXiv:1606.04671*, 2016.

Ozan Sener and Silvio Savarese. Active learning for convolutional neural networks: A core-set approach. *ICLR*, 2018.

Leslie N. Smith. Cyclical learning rates for training neural networks. *WACV*, pp. 464–472, 2017.

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *CVPR*, pp. 1–9, 2015.

Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *CVPR*, pp. 2818–2826, 2016.

Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *AAAI*, 2017.

Russell H Taylor, Arianna Menciassi, Gabor Fichtinger, and Paolo Dario. Medical robotics and computer-integrated surgery. *Springer handbook of robotics*, pp. 1199–1222, 2008.

T. Tieleman and G. Hinton. RMSProp: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural Networks for Machine Learning*, 2012.

Xiaolong Wang, Ross Girshick, Abhinav Gupta, and Kaiming He. Non-local neural networks. *CVPR*, 2018.

Yu-Xiong Wang, Deva Ramanan, and Martial Hebert. Growing a brain: Fine-tuning by increasing model capacity. In *CVPR*, pp. 2471–2480, 2017.

Ashia C Wilson, Rebecca Roelofs, Mitchell Stern, Nati Srebro, and Benjamin Recht. The marginal value of adaptive gradient methods in machine learning. In *NIPS*, pp. 4148–4158. 2017.

Saining Xie and Zhuowen Tu. Holistically-nested edge detection. In *ICCV*, 2015.

Zhichao Yin, Trevor Darrell, and Fisher Yu. Hierarchical discrete distribution decomposition for match density estimation. *CVPR*, 2019.

Hengshuang Zhao, Jianping Shi, Xiaojuan Qi, Xiaogang Wang, and Jiaya Jia. Pyramid scene parsing network. *CVPR*, pp. 6230–6239, 2017.

Martin Zinkevich. Online convex programming and generalized infinitesimal gradient ascent. In *ICML*, pp. 928–936, 2003.

Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning. *ICLR*, 2017.

A BUDGETED TRAINING FOR NEURAL ARCHITECTURE SEARCH

A.1 RANK PREDICTION

In the main text, we list neural architecture search as an application of budgeted training. Due to resource constraint, these methods usually train models with a small budget (10-25 epochs) to evaluate their relative performance (Cao et al., 2019; Cai et al., 2018; Real et al., 2019). Under this setting, *the goal is to rank the performance of different architectures* instead of obtaining the best possible accuracy as in the regular case of budgeted training. Then one could ask the question that whether budgeted training techniques help in better predicting the relative rank. Unfortunately, budgeted training has not been studied or discussed in the neural architecture search literature, it is unknown how well models only trained with 10 epochs can tell the relative performance of the same ones that are trained with 200 epochs. Here we conduct a controlled experiment and show that proper adjustment of learning schedule, specifically the linear schedule, indeed improves the accuracy of rank prediction.

We adapt the code in (Cao et al., 2019) to generate 100 random architectures, which are obtained by random modifications (adding skip connection, removing layer, changing filter numbers) on top of ResNet-18 (He et al., 2017). First, we train these architectures on CIFAR-10 given full budget (200 epochs), following the setting described in Sec 4.1. This produces a relative rank between all pairs of random architectures based on the validation accuracy and this rank is considered as the target to predict given limited budget. Next, every random architecture is trained with various learning schedules under various small budgets. For each schedule and each budget, this generates a complete rank. We treat this rank as the prediction and compare it with the target full-budget rank. The metric we adopt is Kendall's rank correlation coefficient (τ), a standard statistics metric for measuring rank similarity. It is based on counting the inversion pairs in the two ranks and $(\tau + 1)/2$ is approximately the probability of estimating the rank correctly for a pair.

We consider the following schedules: (1) constant, it might be possible that no learning rate schedule is required if only the relative performance is considered. (2) step decay ($\gamma = 0.1$, decay at $p \in \{\frac{1}{3}, \frac{2}{3}\}$), a schedule commonly used both in regular training and neural architecture search (Zoph & Le, 2017; Pham et al., 2018). (3) cosine, a schedule often used in neural architecture search (Cai et al., 2018; Real et al., 2019). (4) linear, our proposed schedule. The results of their rank prediction capability can be seen in Tab 4.

The results suggest that with more budget, we can better estimate the full-budget rank between architectures. And even if only relative performance is considered, learning rate decay should be applied. Specifically, smooth-decaying schedule, such as linear or cosine, are preferred over step decay.

We list some additional details about the experiment. To reduce stochastic noise, each configuration under both the small and full budget is repeated 3 times and the median accuracy is taken. The full-budget model is trained with linear schedule, similar results are expected with other schedules as evidenced by the CIFAR-10 results in the main text (Tab 2). Among the 100 random architectures, 21 cannot be trained, the rest of 79 models have validation accuracy spanning from 0.37 to 0.94, with the distribution mass centered at 0.91. Such skewed and widespread distribution is the typical case in neural architecture search. We remove the 21 models that cannot be trained for our experiments. We take the epoch with the best validation accuracy for each configuration, so the drawback of constant or step decay not having the best model at the very end does not affect this experiment (see Sec 5).

Epoch (Budget)	1 (0.5%)	2 (1%)	10 (5%)	20 (10%)
const	0.3451	0.4595	0.6720	0.6926
step-d2	0.2746	0.3847	0.6651	0.7279
cosine	0.3211	**0.4847**	0.7023	**0.7563**
linear	**0.3409**	0.4348	**0.7398**	0.7351

Table 4: Small-budget and full-budget model rank correlation measured in Kendall's tau. Smooth-decaying schedules like linear and cosine can more accurately predict the true rank of different architectures given limited budget.

Epoch (Budget)	1 (0.5%)	2 (1%)	10 (5%)	20 (10%)
const	0.3892	0.4699	0.6689	0.7061
step-d2	0.4014	0.4780	0.6980	0.7754
cosine	0.4616	0.5498	0.7530	0.8029
linear	**0.4759**	**0.5745**	**0.7652**	**0.8192**

Table 5: Small-budget validation accuracy averaged across random architectures. Linear schedule is the most robust under small budgets.

Epoch (Budget)	1 (0.5%)	2 (1%)	10 (5%)	20 (10%)
const	0.4419	0.5343	0.7550	0.8015
step-d2	0.4590	0.5455	0.7894	0.8848
cosine	0.5326	0.6265	0.8615	0.9087
linear	**0.5431**	**0.6626**	**0.8644**	**0.9305**

Table 6: Tab 5 normalized by the full-budget accuracy and then averaged across architectures. Linear schedule achieves solutions closer to their full-budget performance than the rest of schedules under small budgets.

A.2 Budgeted Performance Across Architectures

To reinforce our claim that linear schedule generalizes across different settings, we compare budgeted performance of various schedules on random architectures generated in the previous section. We present two versions of the results. The first is to directly average the validation accuracy of different architecture with each schedule and under each budget (Tab 5). The second is to normalize by dividing the budgeted accuracy by the full-budget accuracy of the same architecture and then average across different architectures. The second version assumes all architectures enjoy equal weighting. Under both cases, linear schedule is the most robust schedule across architectures under various budgets.

B Equivalent Learning Rate For AMSGrad

In Sec 5, we use equivalent learning rate to compare AMSGrad (Reddi et al., 2018) with momentum SGD. Here we present the derivation for the equivalent learning rate $\widetilde{\beta}_t$.

Let η_1, η_2 and ϵ be hyper-parameters, then the momentum SGD update rule is:

$$m_t = \eta_1 m_{t-1} + (1-\eta_1) g_t, \tag{3}$$

$$w_t = w_{t-1} - \alpha_0^{(1)} \beta_t m_t, \tag{4}$$

while the AMSGrad update rule is:

$$m_t = \eta_1 m_{t-1} + (1-\eta_1) g_t, \tag{5}$$

$$v_t = \eta_2 v_{t-1} + (1-\eta_2) g_t^2, \tag{6}$$

$$\hat{m}_t = \frac{m_t}{1-\eta_1^t}, \tag{7}$$

$$\hat{v}_t = \frac{v_t}{1-\eta_2^t}, \tag{8}$$

$$\hat{v}_t^{\max} = \max(\hat{v}_t^{\max}, \hat{v}_t) \tag{9}$$

$$w_t = w_{t-1} - \alpha_0^{(2)} \frac{\hat{m}_t}{\sqrt{\hat{v}_t^{\max}} + \epsilon}. \tag{10}$$

Comparing equation 4 with 10, we obtain the equivalent learning rate:

$$\widetilde{\beta}_t = \frac{\alpha_0^{(2)}}{\alpha_0^{(1)}} \frac{1}{(1-\eta_1^t)(\sqrt{\hat{v}_t^{\max}} + \epsilon)}, \tag{11}$$

Budget	1%	5%	10%	25%	50%	100%
Subset	.3834	.6446	.7848	.8586	.9234	N/A
Full	**.5544**	**.8328**	**.9042**	**.9338**	**.9464**	**.9534**

Table 7: Comparison with offline data subsampling. "Subset" meets the budget constraint by randomly subsample the dataset prior to training, while "full" uses all the data, but restricting the number of iterations. Note that budget-aware schedule is used for "full".

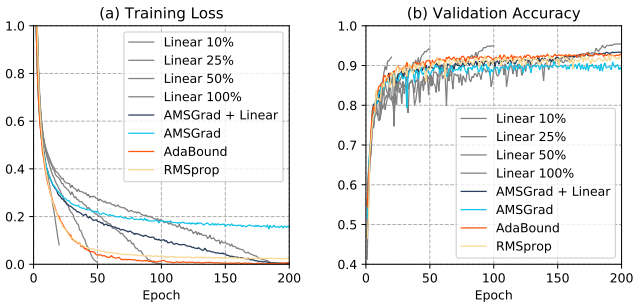
Note that the above equation holds per each weight. For Fig 4a, we take the median across all dimensions as a scalar summary since it is a skewed distribution. The mean appears to be even larger and shares the same trend as the median. In our experiments, we use the default hyper-parameters (which also turn out to have the best validation accuracy): $\alpha_0^{(1)} = 0.1$, $\alpha_0^{(2)} = 0.001$, $\eta_1 = 0.9$, $\eta_2 = 0.99$ and $\epsilon = 10^{-8}$.

C Data Subsampling

Data subsampling is a straight-forward strategy for budgeted training and can be realized in several different ways. In our work, we limit the number of iterations to meet the budget constraint and this effectively limits the number of data points seen during the training process. An alternative is to construct a subsampled dataset offline, but keep the same number of training iterations. Such construction can be done by random sampling, which might be the most effective strategy for *i.i.d* [5] dataset. We show in Tab 7 that even our baseline budge-aware step decay, together with a limitation on the iterations, can significantly outperform this offline strategy. For the subset setting, we use the off-the-shelf step decay (step-d2) while for the full set setting, we use the same step decay but with BAC applied (Sec 3.1). For detailed setup, we follow Sec 4.1, of the main text and take the median of 3 runs.

Of course, more complicated subset construction methods exist, such as *core-set* construction (Bachem et al., 2017). However, such methods usually requires a feature summary of each data point and the computation of pairwise distance, making such methods unsuitable for extremely large dataset. In addition, note that our subsampling experiment is conducted on CIFAR-10, a well-constructed and balanced dataset, making smarter subsampling methods less advantageous. Consequently, the result in Tab 7 can as well provides a reasonable estimate for other complicated subsampling methods.

D Additional Comparison with Adaptive Learning Rates



Method	Val Accu
RMSprop	.9258
AdaBound	.9306
AMSGrad	.9113
AMSGrad + Linear	.9340
SGD + Linear 10%	.9218
SGD + Linear 25%	.9412
SGD + Linear 50%	.9546
SGD + Linear 100%	**.9562**

Figure 6: Comparison between budget-aware linear schedule and adaptive learning rate methods on CIFAR-10. We see while adaptive learning rate methods appear to descent faster than full budget linear schedule, at each given budget, they are surpassed by the corresponding linear schedule.

In the main text we compare linear schedule with AMSGrad (Reddi et al., 2018) (the improved version over Adam (Kingma & Ba, 2015)), we further include the classical method RMSprop (Tieleman

[5] independent and identically distributed

Budget	1%	5%	10%	25%	50%	100%
step	.4944	.6332	.6796	.7242	.7465	**.7646**
poly	**.5474**	**.6751**	**.7117**	**.7411**	**.7575**	.7601
linear	.5414	.6657	.7079	.7400	.7560	.7629

Table 8: Comparison with step decay and the off-the-shelf poly schedule on Cityscapes (Cordts et al., 2016) using PSPNet (Zhao et al., 2017). Both poly and linear significantly outperforms step decay given limited budget. Poly and linear are similar smooth-decaying schedules (Fig 2) and thus have similar performance. The exact rank differs from task to task. Same as other main experiments, the median of 3 runs is reported.

& Hinton, 2012) and the more recent AdaBound (Luo et al., 2019). We tune these adaptive methods for CIFAR-10 and summarize the results in Fig 6. We observe the similar conclusion that budget-aware linear schedule outperforms adaptive methods for all given budgets.

Like SGD, those adaptive learning rate methods also takes input a parameter of base learning rate, which can also be annealed using an existing schedule. Although it is unclear why one needs to anneal an adaptive methods, we find that it in facts boosts the performance ("AMSGrad + Linear" in Fig 6).

E Experiments on Cityscapes (Semantic Segmentation)

Cityscapes (Cordts et al., 2016) is a dataset commonly used for evaluating semantic segmentation algorithms. It contains high quality pixel-level annotations of 5k images in urban scenarios. The default evaluation metric is the mIoU (averaged across class) of the output segmentation map. We use state-of-the-art model PSPNet (Zhao et al., 2017) with a ResNet-50 backbone and the full budget is 400 epochs as in standard set up. The mIoU of the best epoch is reported. Interestingly, unlike other tasks in this series, this model by default uses the poly (Jia et al., 2014) schedule. For complete evaluation, we also add a step decay that is the same in our ImageNet experiment.

In addition, we find linear schedule shares similar performance with the off-the-shelf poly schedule on Cityscapes (Tab 8). Given the similarity of poly and linear (Fig 2), and the opposite results on CIFAR-10 and Cityscapes, it is inconclusive that one is strictly better than the other within the smooth-decaying family. However, these smooth-decaying methods both outperform step decay given limited budgets.

F Additional Comparison with SGDR

This section provides additional evaluation to show that learning rate restart produces worse results than our proposed budgeted training techniques under budgeted setting. In (Loshchilov & Hutter,

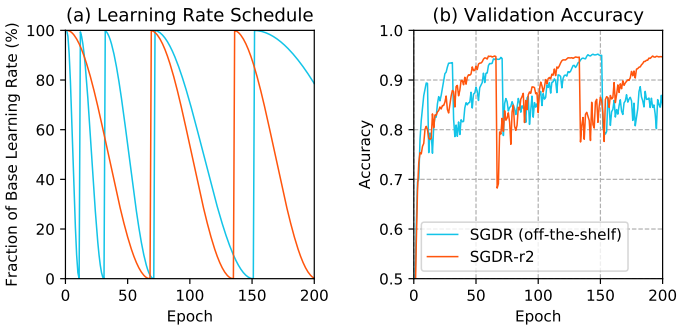


Figure 7: One issue with off-the-shelf SGDR ($T_0 = 10$, $T_{\text{mult}} = 2$) is that it is not budget-aware and might end at a poor solution. We convert it to a budget aware schedule by setting it to restart n times at even intervals across the budget and $n = 2$ is shown here (SGDR-r2).

Epoch	30	50	150
SGDR	.9320	.9458	.9510
linear	**.9350**	**.9506**	**.9532**

Table 9: Comparison with off-the-shelf SGDR at the end of each period after the first restart.

Budget	1%	5%	10%	25%	50%	100%
SGDR-r1	.5002	.7908	.8794	.9250	.9380	.9488
SGDR-r2	.4710	.7888	.8738	.9216	.9412	.9502
linear	**.6654**	**.8920**	**.9218**	**.9412**	**.9546**	**.9562**

Table 10: Comparison with SGDR under budget-aware setting. "SGDR-r1" refers to restarting learning rate once at midpoint of the training progress, and "SGDR-r2" refers to restarting twice at even interval.

2016), both a new form of decay (cosine) and the technique of learning rate restart are proposed. To avoid confusion, we use "cosine schedule", or just "cosine", to refer to the form of decay and SGDR to a schedule of periodical cosine decays. The comparison with cosine schedule is already included in the main text. Here we focus on evaluating the periodical schedule. SGDR requires two parameters to specify the periods: T_0, the length of the first period; T_{mult}, where i-th period has length $T_i = T_0 T_{\text{mult}}^{i-1}$. In Fig 7, we plot the off-the-shelf SGDR schedule with $T_0 = 10$ (epoch), $T_{\text{mult}} = 2$. The validation accuracy plot (on the right) shows that it might end at a very poor solution (0.8460) since it is not budget-aware. Therefore, we consider two settings to compare linear schedule with SGDR. The first is to compare only at the end of each period of SGDR, where budgeted convergence is observed. The second is to convert SGDR into a budget-aware schedule by setting the schedule to restart n times at even intervals across the budget. The results under the first and second setting is shown in Tab 9 and Tab 10 respectively. Under both budget-aware and budget-unaware setting, linear schedule outperforms SGDR. For detailed setup, we follow Sec 4.1, of the main text and take the median of 3 runs.

G Additional Illustrations

In Sec 5, we refer to validation accuracy curve for training on CIFAR-10, which we provide here in Fig 8.

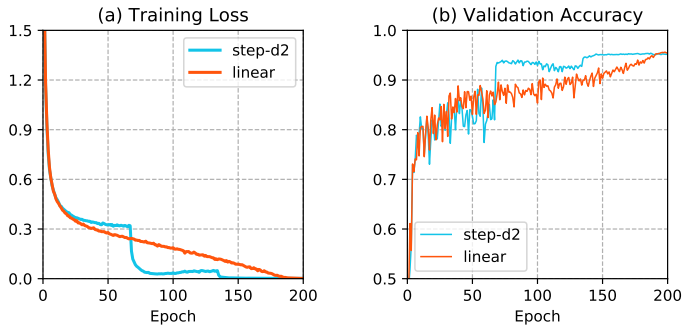


Figure 8: Training loss and validation accuracy for training ResNet-18 on CIFAR-10 using step decay and linear schedule. No generalization gap is observed when we only modify learning rate schedule. This figure provides details for the discussion of "don't waste budget".

Budget	1%	5%	10%	25%	50%	100%
const	.5748 ± .0337	.7989 ± .0093	.8350 ± .0122	.8658 ± .0007	.8723 ± .0044	.8767 ± .0066
exp .95	.4834 ± .0125	.7575 ± .0053	.8567 ± .0027	.9147 ± .0030	.9295 ± .0006	.9468 ± .0021
exp .97	.5467 ± .0202	.8348 ± .0016	.8936 ± .0030	.9294 ± .0024	.9413 ± .0015	.9551 ± .0004
exp .99	.6069 ± .0219	.8557 ± .0037	.9013 ± .0036	.9227 ± .0033	.9268 ± .0026	.9310 ± .0023
step-d1	.5853 ± .0134	.8643 ± .0027	.9063 ± .0023	.9307 ± .0020	.9423 ± .0027	.9426 ± .0031
step-d2	.5487 ± .0156	.8342 ± .0052	.9043 ± .0034	.9319 ± .0037	.9461 ± .0019	.9529 ± .0009
step-d3	.4879 ± .0036	.7929 ± .0061	.8864 ± .0027	.9259 ± .0006	.9437 ± .0001	.9527 ± .0019
htd	.6450 ± .0070	.8899 ± .0043	.9219 ± .0014	.9449 ± .0031	.9520 ± .0023	.9554 ± .0013
cosine	.6343 ± .0080	.8851 ± .0024	.9223 ± .0024	.9432 ± .0024	.9520 ± .0026	.9552 ± .0021
poly	.6595 ± .0086	.8905 ± .0017	.9247 ± .0008	.9421 ± .0019	.9494 ± .0034	.9540 ± .0012
linear	.6617 ± .0079	.8915 ± .0011	.9217 ± .0028	.9412 ± .0018	.9537 ± .0020	.9563 ± .0009

Table 11: Mean and standard deviation for Fig 2 — various learning rate schedules on CIFAR-10. This table shows that the improvement made by smooth-decaying schedules is beyond the noise level, while the relative improvement within the smooth-decaying schedules is in general within the noise level.

Budget	1%	5%	10%	25%	50%	100%
Image classification on ImageNet with ResNet						
step	.2039 ± .0029	.5194 ± .0048	.5951 ± .0021	.6558 ± .0018	.6796 ± .0008	**.6934** ± .0018
linear	**.3063** ± .0029	**.5726** ± .0048	**.6232** ± .0021	**.6634** ± .0018	**.6818** ± .0008	.6933 ± .0018
Object detection on COCO with Mask-RCNN						
step	.0486 ± .0024	.2003 ± .0008	.2541 ± .0005	.3149 ± .0015	.3530 ± .0005	.3767 ± .0009
linear	**.0513** ± .0042	**.2090** ± .0016	**.2626** ± .0008	**.3222** ± .0014	**.3572** ± .0003	**.3795** ± .0012
Instance segmentation on COCO with Mask-RCNN						
step	.0487 ± .0029	.1925 ± .0004	.2388 ± .0007	.2907 ± .0003	.3202 ± .0009	.3395 ± .0009
linear	**.0507** ± .0040	**.1986** ± .0012	**.2457** ± .0007	**.2942** ± .0002	**.3242** ± .0005	**.3396** ± .0009
Video classification on Kinetics with I3D						
step	.2941 ± .0028	.4981 ± .0029	.5674 ± .0013	.6459 ± .0023	.6870 ± .0025	.7134 ± .0021
linear	**.3286** ± .0042	**.5297** ± .0014	**.5967** ± .0030	**.6634** ± .0020	**.6995** ± .0011	**.7223** ± .0031

Table 12: Mean and standard deviation for Tab 2 — linear schedule against off-the-shelf step decay on various vision benchmarks. This table shows that the improvement made by budget-aware linear schedule is beyond the noise level.

H Learning Rates in Convex Optimization

For convex cost surfaces, constant learning rates are guaranteed to converge when less or equal than $1/L$, where L is the Lipschitz constant for the gradient of the cost function ∇F (Bottou et al., 2018). Another well-known result ensures convergence for sequences that decay neither too fast nor too slow (Robbins & Monro, 1951): $\sum_{t=1}^{\infty} \alpha_t = \infty, \sum_{t=1}^{\infty} \alpha_t^2 < \infty$. One common such instance in convex optimization is $\alpha_t = \alpha_0/t$. For non-convex problems, similar results hold for convergence to a local minimum (Bottou et al., 2018). Unfortunately, there does not exist a theory for learning rate schedules in the context of general non-convex optimization.

I Variance in our experiments

Neural network training is a stochastic process. In the main text, we report the median of 3 independent runs. Here we also report the mean and the standard deviations of the same 3 runs in Tab 11 (CIFAR10, corresponding to Fig 2) and Tab 12 (vision benchmarks, corresponding to Tab 2)

J Full Gradient Norm and the Weight Norm

In Fig 3, 4a, 4b, we plot the full gradient norm of the cross-entropy loss, excluding the regularization loss. In fact, we use an L2-regularization (weight decay) of 0.0004 for these experiments.

Batch Size	Schedule	20%	50%	100%
64	step-d2	.9436 ± .0037	.9505 ± .0009	.9519 ± .0009
64	linear	**.9473** ± .0021	**.9511** ± .0008	**.9526** ± .0020
256	step-d2	.8939 ± .0027	.9291 ± .0021	.9431 ± .0008
256	linear	**.9143** ± .0018	**.9415** ± .0038	**.9484** ± .0013
1024	step-d2	.5851 ± .0460	.7703 ± .0121	.8805 ± .0007
1024	linear	**.7415** ± .0141	**.8553** ± .0023	**.8992** ± .0042

Table 13: Comparison between linear and step decay with different batch sizes. We can see that even when we vary the batch size, linear schedule outperforms step decay.

Initial LR	0.001	0.1	1	10
step-d2	.9152 ± .0024	.9529 ± .0009	.8869 ± .0065	N/A
linear	**.9167** ± .0023	**.9563** ± .0009	**.8967** ± .0034	N/A

Table 14: Comparison between linear and step decay with different initial learning rate under full budget setting. On one hand, we see that linear schedule outperforms step decay under various initial learning rate. On the other hand, we see that initial learning rate is still a very important hyper-parameter that needs to be tuned even with budget-aware, smooth-decaying schedules.

K More ablation studies

Here we explore variations of batch size (Tab 13) and initial learning rate (Tab 14). Our definition of budget is the number of examples seen during training. So when the batch size increases, the number of iterations decreases. For example, on CIFAR-10, the full budget is training with batch size 128 for 200 epochs. If we train with batch size 1024 for 20% of the budget, that means training for 5 epochs.

L Additional Implementation Details

Image classification on ImageNet. We adapt both the network architecture (ResNet-18) and the data loader from the open source PyTorch ImageNet example[6]. The base learning rate used is 0.1

[6] `https://github.com/pytorch/examples/tree/master/imagenet`. PyTorch version 0.4.1.

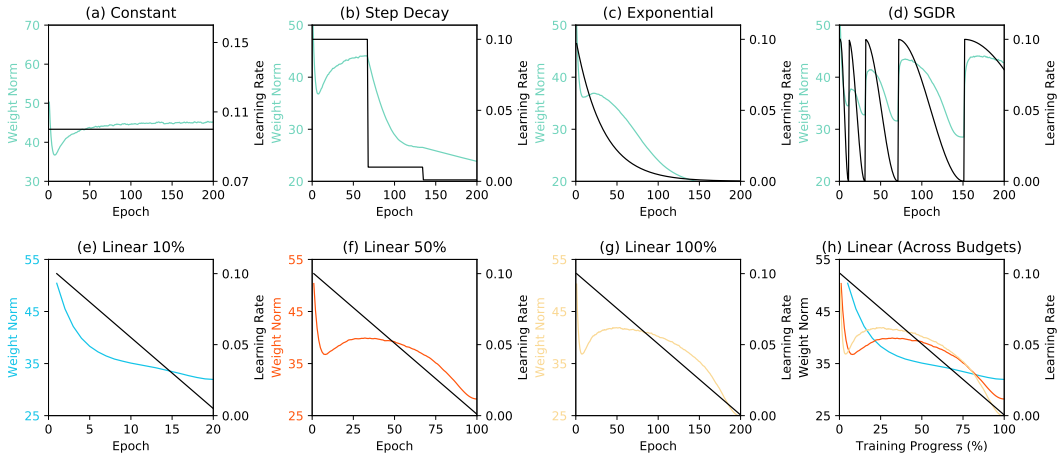


Figure 9: Weight norm changes as learning rate changes with standard weight decay of 0.005.

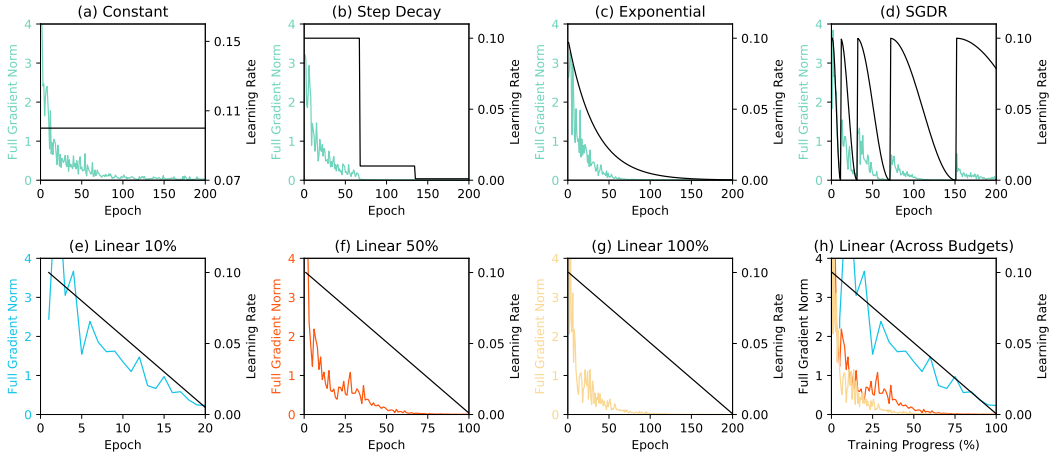


Figure 10: A similar but much weaker correlation between the full gradient norm and learning rate with the absence of weight decay.

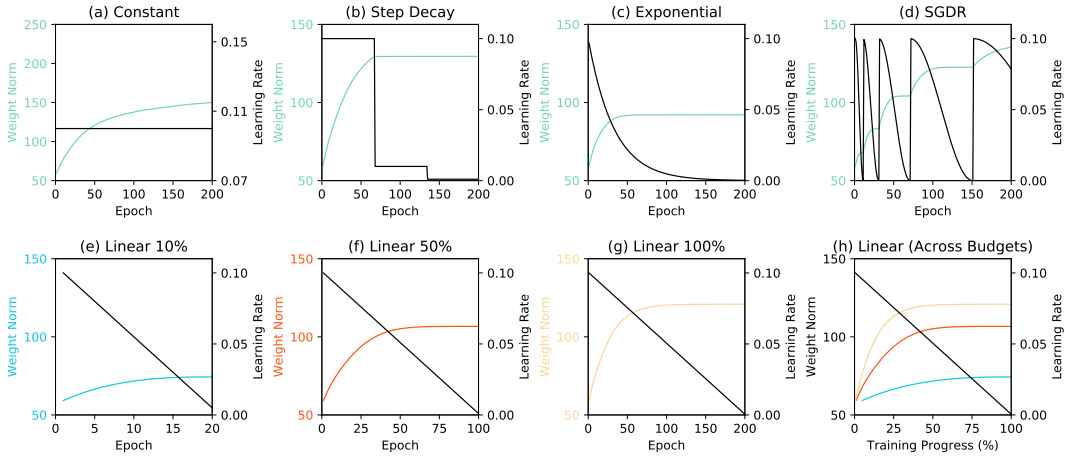


Figure 11: Weight norm grows almost independently as learning rate changes with the absence of weight decay.

and weight decay 5×10^{-4}. We train using 4 GPUs with asynchronous batch normalization and batch size 128.

Video classification on Kinetics with I3D. The 400-category version of the dataset is used in the evaluation. We use an open source codebase[7] that has training and data processing code publicly available. Note that the codebase implements a variant of standard I3D (Carreira & Zisserman, 2017) that has ResNet as the backbone. We follow the configuration of `run_i3d_baseline_300k_4gpu.sh`, which specifies a base learning rate 0.005 and a weight decay 10^{-4}. Only learning rate schedule is modified in our experiments. We train using 4 GPUs with asynchronous batch normalization and batch size 32.

Object detection and instance segmentation on MS COCO. We use the open source implementation of Mask R-CNN[8], which is a PyTorch re-implementation of the official codebase Detectron in the Caffe 2 framework. We only modify the part of the code for learning rate schedule. The codebase sets base learning rate to 0.02 and weight decay 10^{-4}. We train with 8 GPUs (batch size 16) and keep the built-in learning rate warm up mechanism, which is an implementation technique that increases learning rate for 0.5k iterations and is intended for stabilizing the initial phase of multi-

[7] `https://github.com/facebookresearch/video-nonlocal-net`. Caffe 2 version 0.8.1.
[8] `https://github.com/roytseng-tw/Detectron.pytorch`. PyTorch version 0.4.1.

GPU training (Goyal et al., 2017). The 0.5k iterations are kept fixed for all budgets and learning rate decay is applied to the rest of the training progress.

Semantic segmentation on Cityscapes. We adapt a PyTorch codebase obtained from correspondence with the authors of PSPNet. The base learning rate is set to 0.01 with weight decay 10^{-4}. The training time augmentation includes random resize, crop, rotation, horizontal flip and Gaussian blur. We use patch-based testing time augmentation, which cuts the input image to patches of 713×713 and processes each patch independently and then tiles the patches to form a single output. For overlapped regions, the average logits of two patches are taken. We train using 4 GPUs with *synchronous* batch normalization and batch size 12.