REX: REVISITING BUDGETED TRAINING WITH AN IMPROVED SCHEDULE

John Chen [1] **Cameron Wolfe** [1] **Anastasios Kyrillidis** [1]

ABSTRACT

Deep learning practitioners often operate on a computational and monetary budget. Thus, it is critical to design optimization algorithms that perform well under any budget. The linear learning rate schedule is considered the best budget-aware schedule (Li et al., 2020), as it outperforms most other schedules in the low budget regime. On the other hand, learning rate schedules –such as the `30-60-90` step schedule– are known to achieve high performance when the model can be trained for many epochs. Yet, it is often not known a priori whether one's budget will be large or small; thus, the optimal choice of learning rate schedule is made on a case-by-case basis. In this paper, we frame the learning rate schedule selection problem as a combination of i) selecting a profile (i.e., the continuous function that models the learning rate schedule), and ii) choosing a sampling rate (i.e., how frequently the learning rate is updated/sampled from this profile). We propose a novel profile and sampling rate combination called the Reflected Exponential (REX) schedule, which we evaluate across seven different experimental settings with both SGD and Adam optimizers. REX outperforms the linear schedule in the low budget regime, while matching or exceeding the performance of several state-of-the-art learning rate schedules (linear, step, exponential, cosine, step decay on plateau, and OneCycle) in both high and low budget regimes. Furthermore, REX requires no added computation, storage, or hyperparameters.

1 INTRODUCTION

While hardware has consistently improved (Sze et al., 2017; Shawahna et al., 2019), the cost of training deep neural networks (DNNs) has continued to increase due to growth in the size of models and datasets (Krizhevsky et al., 2012; Devlin et al., 2019; et al., 2020a; Chen et al., 2020b). One key component of the cost is the need to tune the hyperparameters of the model (Yang & Shami, 2020). Outside of the largest companies in the field, most practitioners have to trade-off the number of epochs with the number of experimental trials. Whilst the community has generally agreed that, for example, 90 epochs is a reasonable training length for a ResNet-50 architecture on ImageNet (He et al., 2016; Huang et al., 2017; Zagoruyko & Komodakis, 2016), there simply may not be sufficient monetary budget to perform such extensive training for certain projects. Further, it is generally not easy to predict the number of epochs required to maximize the performance of the model apriori, particularly if the input data may be continually changing. *Thus, it is important to consider the optimization of DNNs for a diverse range of budgets.*

Stochastic Gradient Descent (SGD) with momentum and Adam are two of the most widely used optimizers for DNNs (He et al., 2016; Huang et al., 2017; Zagoruyko & Komodakis, 2016; Devlin et al., 2019; et al., 2020a; Redmon & Farhadi, 2018). Whether the task is image classification, object detection, or fine-tuning in natural language processing, both optimizers must be combined with some form of learning rate decay to achieve good performance (He et al., 2016; Huang et al., 2017; Zagoruyko & Komodakis, 2016; Devlin et al., 2019; et al., 2020a; Redmon & Farhadi, 2018) (see Tables 4-11). The aforementioned tasks are arguably the most widely used applications of deep learning.[1]

The learning rate schedule is particularly important in the budgeted training setting. Moreover, of the widely used schedules, the best learning rate schedule for a small number of epochs is generally not the best for a large number of epochs (see Tables 4-11). This is a significant challenge, since it is difficult to know apriori if the current budget lies in the high or low budget regime. This raises two questions: *Can we close the budget-induced gap in the performance of existing learning rate schedules? And, if this is not possible, is there a learning rate schedule that performs well in both low and high budget regimes?*

We answer both questions through a novel lens.

[*]Equal contribution [1]Rice University, Houston, USA. Correspondence to: John Chen <johnchen@rice.edu>.

Proceedings of the 5^{th} MLSys Conference, Santa Clara, CA, USA, 2022.

[1]There are some cases in which learning rate decay is not always useful, such as for Generative Adversarial Networks (Goodfellow et al., 2014; Arjovsky et al., 2017), but this is generally a small proportion of all deep learning activities.

Table 1. Performance of different schedules, ranked according to the % of Top-1 or Top-3 finishes, out of a total of 28 experiments. Top-1 (Top-3) refers to the best (best-3) performance for a particular model/dataset/base optimizer/epoch setting. Low (high) budget includes 1%, 5%, and 10% (25%, 50%, and 100%) of the full epochs. The Decay on Plateau variant is aggregated into the Step Schedule method where we take the max performance for each setting.

	Low budget (<25%)		High budget (≥25%)		Overall	
Method	Top-1	Top-3	Top-1	Top-3	Top-1	Top-3
None	0%	0%	2%	10%	1%	5%
Exp decay (Abadi et al., 2015)	5%	7%	5%	14%	5%	11%
OneCycle (Smith, 2018)	15%	49%	12%	40%	13%	45%
Linear Schedule (Abadi et al., 2015)	10%	78%	12%	62%	11%	70%
Step Schedule (He et al., 2016)	2%	12%	7%	38%	5%	25%
Cosine Schedule (Loshchilov & Hutter, 2017b)	2%	66%	10%	62%	6%	64%
REX	**73%**	**95%**	**67%**	**88%**	**70%**	**92%**

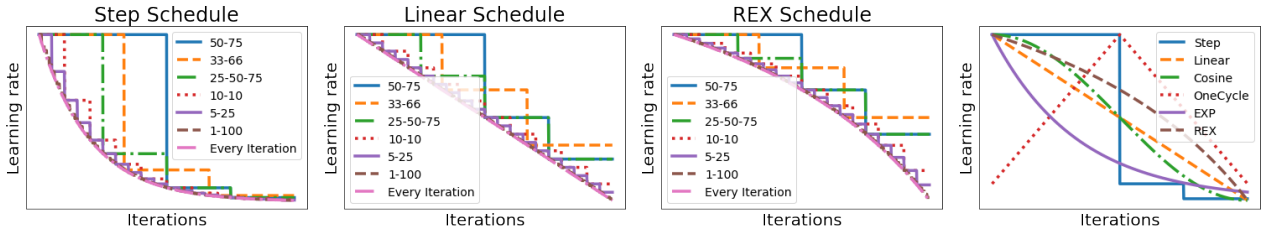

Figure 1. Popular schedules with various sampling rates. `50-75` refers to sampling once at 50% and 75% of total epochs. Similarly for `33-66` and `25-50-75`. `10-10` refers to sampling once every 10% of total epochs. Similarly for `5-25` and `1-100`. Every iteration is the maximum sampling rate. Left: Step schedule. Left Middle: Linear Schedule. Right Middle: REX Schedule. Right: Schedules with their usual sampling rate.

We decompose the problem of selecting a learning rate schedule as a two-part process of i) selecting a profile and ii) selecting a sampling rate. The *profile* is the function that models the learning rate schedule, and the *sampling rate* is how frequently the learning rate is updated, based on this profile. In this view, we i) analyze existing schedules, ii) propose a novel profile and sampling rate combination, and iii) benchmark the performance of numerous schedules. We also demonstrate it is possible to boost the performance of existing learning rate schedules by introducing a hyperparameter that delays the commencement of the decay schedule. However, because adding an extra hyperparameter is prohibitive in the budgeted setting, we also propose a new schedule, REX, which performs at a state-of-the-art level for both low and high budgets across a large variety of settings without the extra hyperparameter tuning.

Specifically, our contributions are as follows:

- We pose learning rate schedules as the combination of a profile and a sampling rate and identify that there is no optimal profile for all sampling rates. Namely, we show that no existing, popular learning rate schedule achieves state-of-the-art performance in both high and low budget regimes.
- We propose a new profile and sampling rate combination. We find that carefully tuning the start of the learning rate decay for existing schedules can result in significant performance improvements in both high and low budget regimes. However, this introduces an extra hyperparameter, which is prohibitive for budget-limited practitioners. Our proposed schedule can be understood as an interpolation between the linear schedule and the delayed variants.
- Our proposed schedule, REX, is based on observations of the above, and we validate its state-of-the-art performance across seven settings, including image classification, object detection, and natural language processing.

Our goal is to introduce an easy-to-use, state-of-the-art learning rate schedule with no extra hyperparameters that performs well in all budget regimes and can be easily implemented and adopted.

2 Related Works

There have been many works related to tuning the learning rate. There is a connection between learning rate and momentum (Yuan et al., 2016), and there are methods which alter the momentum (Sutskever et al., 2013; Zhang

Table 2. We demonstrate learning rate schedules and sampling rates on `RN20-CIFAR10-SGDM` (Top) and `RN38-CIFAR10-SGDM` (Bottom) (He et al., 2016), holding the learning rate constant. There is no best profile for all sampling rates. Each profile excels at one end of the spectrum. `50-75` (He et al., 2016) refers to sampling once at 50% and 75% of total epochs. Similarly for `33-66` and `25-50-75`. `10-10` refers to sampling once every 10% of total epochs. Similarly for `5-25` and `1-100`. Every iteration is the maximum sampling rate.

RN20-CIFAR10-SGDM	15 Epochs			75 Epochs			300 Epochs		
Sampling Rate	Step	Linear	REX	Step	Linear	REX	Step	Linear	REX
50-75	14.48	16.96	20.79	9.44	12.42	18.05	**7.32**	10.15	12.41
33-66	17.89	25.80	24.45	9.72	13.38	15.98	7.93	11.90	11.43
25-50-75	16.52	18.77	26.13	9.73	12.31	12.59	8.46	8.26	12.31
10-10	17.98	16.35	16.48	10.41	9.40	11.17	8.67	8.26	8.24
5-25	18.87	13.83	15.17	9.79	8.94	9.22	8.85	8.24	8.50
1-100	18.53	13.91	**13.34**	10.61	**8.72**	**8.60**	9.20	7.97	7.74
Every Iteration	19.19	**13.09**	**12.86**	9.97	8.89	**8.37**	9.24	**7.62**	**7.52**

RN38-CIFAR10-SGDM	15 Epochs			75 Epochs			300 Epochs		
Sampling Rate	Step	Linear	REX	Step	Linear	REX	Step	Linear	REX
50-75	13.57	17.31	18.47	7.59	12.89	14.38	6.66	10.07	9.37
33-66	14.96	19.16	18.71	7.74	13.64	17.57	6.70	11.53	11.30
25-50-75	15.69	14.18	19.77	7.99	9.10	15.07	6.73	7.59	8.44
10-10	16.58	13.34	14.46	7.87	8.33	9.75	7.60	6.48	6.50
5-25	17.16	12.63	**11.71**	8.40	7.42	7.13	8.79	6.18	6.41
1-100	17.20	11.93	**11.13**	8.54	**7.06**	7.17	9.11	**6.12**	6.17
Every Iteration	17.97	12.11	**10.95**	8.72	**7.10**	**6.86**	9.31	**5.89**	**6.09**

& Mitliagkas, 2017; Odonoghue & Candes, 2015; Lucas et al., 2018; Chen et al., 2020a). There is also a connection between learning rate and batch sizes (Smith et al., 2017; You et al., 2017; Goyal et al., 2018). The most popular learning rate tuning mechanisms fall into two categories: Automatically tuning the learning rate on a per-weight basis and decaying the learning rate globally.

Many adaptive learning rate optimizers have been proposed. Modern learning rate adaptive methods began with AdaGrad (Duchi et al., 2011), which was shown to have good convergence properties, especially in the sparse gradient setting. AdaDelta (Zeiler, 2012) was proposed to fix a units issue with AdaGrad. RMSprop (Hinton et al., 2012) employed a running estimate of the second moment to resolve the strictly decreasing learning rate of AdaGrad. The most popular adaptive learning rate optimizer is Adam (Kingma & Ba, 2014) and its variants (Loshchilov & Hutter, 2017a; Liu et al., 2020). *Yet, in practice, adaptive learning rate algorithms perform the best when coupled with a learning rate schedule* (Devlin et al., 2019; Liu et al., 2020).

In deep learning, the step schedule was widely used in early computer vision work (Krizhevsky et al., 2012; He et al., 2016; Huang et al., 2017). This was often combined with SGD with Momentum to achieve state-of-the-art results (He et al., 2016; Zagoruyko & Komodakis, 2016; Huang et al., 2017; Redmon et al., 2016). In Natural Language Processing, AdamW (Loshchilov & Hutter, 2017a) is often paired with a cosine or linear learning rate decay for training and fine-tuning transformers (et al., 2020b).

The aforementioned schedules are widely available and implemented in the most popular software (et al., 2020b; Abadi et al., 2015; Paszke et al., 2017), in addition to the exponential decay schedule, OneCycle (Smith, 2018), cosine decay with restarts (Loshchilov & Hutter, 2017b) and others (Smith, 2017).

While some schedules may be preferred for achieving state-of-the-art results, it has been suggested that the linear schedule is most suitable for the low budget scenario (Li et al., 2020), which may be of more relevance to practitioners.

3 Budgeted Training: Profiles and Sampling Rates

Challenges in adapting learning rate schedules to the budgeted setting. The primary hyperparameter in DNN optimization is the initial learning rate. While good heuristics often exist for tuning common hyperparameters, such as setting momentum $\beta = 0.9$ or setting a `30-60-90` learning rate schedule (Zagoruyko & Komodakis, 2016; Huang et al., 2017; Hu et al., 2017), the initial learning rate remains to be

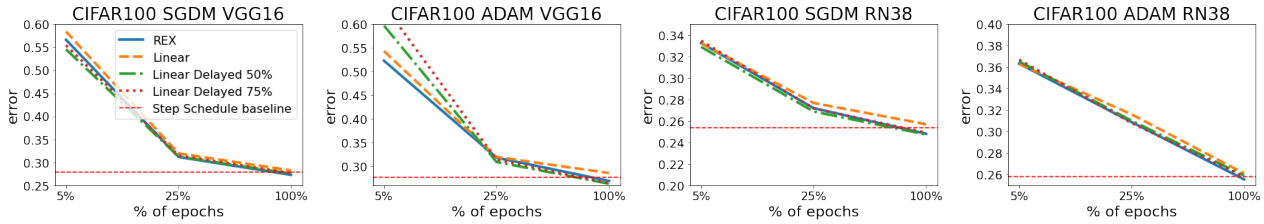

Figure 2. REX, linear, and delayed linear schedules. Left: `VGG16-CIFAR100-SGDM`. Left Middle: `VGG16-CIFAR100-ADAM`. Right Middle: `RN38-CIFAR100-SGDM`. Right: `RN38-CIFAR100-ADAM`. The red dashed line represents the error of the step schedule for that setting trained with 100% of the epochs. Linear Delayed X% refers to delaying the linear decay till X% of the total epochs have passed, before decaying linearly to 0. For example, in the left-middle plot, for small % of epochs, REX outperforms the linear schedule, which outperforms the delayed variants. However, for large epochs, the linear schedule is unable to achieve the state-of-the-art performance of the step schedule, while REX and the delayed linear schedules are able to surpass the step schedule.

Table 3. Summary of experimental settings.

Experiment short name	Model	Dataset	Maximum Epochs
`RN20-CIFAR10`	ResNet20	CIFAR10	300 (He et al., 2016)
`RN50-IMAGENET`	ResNet50	ImageNet	90 (Huang et al., 2017)
`VGG16-CIFAR100`	VGG-16	CIFAR100	300 (He et al., 2016)
`WRN-STL10`	Wide ResNet 16-8	STL10	200 (Chang et al., 2017)
`VAE-MNIST`	VAE	MNIST	200 (Yeung et al., 2017)
`YOLO-VOC`	YOLOv3	Pascal VOC	50 (Tripathi et al., 2016)
$\texttt{BERT}_{BASE}\texttt{-GLUE}$	BERT (Pre-trained)	GLUE (9 tasks)	3 (Devlin et al., 2019)

tuned. However, in the budgeted training setting, the learning rate schedule turns into a hyperparameter. Adapting, for example, the `30-60-90` rule for Image Classification or Object Detection is not straightforward, and naively following the same rules for a smaller number of epochs results in sub-optimal results (see Step Schedule in low epoch settings in Tables 4-11). Additionally, following the `50-75` rule (He et al., 2016) on `RN20-CIFAR10` for a training budget that is 1% of the usual total epochs can result a 5% absolute error gap with the best-performing schedule. We assume that, in the budgeted training setting, the number of epochs is still pre-defined, but can be significantly less than the usual total epochs.

Profiles and sampling rates. To formalize the process of identifying a good learning rate schedule, we decompose the learning rate schedule as a combination of a profile curve and a sampling rate on that curve. The *profile* is the function that models the learning rate schedule and dictates the general curve of the learning rate schedule. In most –but not all (Li & Arora, 2020)– applications, this function starts at a high initial value and ends near zero. The *sampling rate* is how frequently the learning rate is updated and dictates the smoothness of the curve. At one extreme, the linear learning rate schedule, and many others, samples from the profile at each iteration, and at the other extreme the step learning rate schedules samples only twice or thrice across the entire training procedure. For example, the `50-75` step schedule can be approximated as sampling twice from a particular, exponentially-decaying profile. See Figure 1 for some examples of schedules with their associated profile and sampling rates.

Lack of an optimal profile. While there may be limited motivation to pick a particular sampling rate, this introduces an interesting question: *Does there exist an optimal profile for all reasonable sampling rates?* In Table 2, we benchmark three profiles: i) the `50-75` step schedule (He et al., 2016) approximated as a tuned exponentially decaying profile ; ii) the linear profile (Abadi et al., 2015; Paszke et al., 2017); and, iii) the REX profile proposed in this paper (to be defined in the next subsection). These three profiles represent smoothly-decaying learning rate schedules with varying curvatures. We find that different profiles perform best for different sampling rates. *The approximated Step schedule profile performs best with low sampling rates, while the linear and REX profiles perform best with high sampling rates.* Furthermore, *the approximated Step schedule profile performs worst for a small and medium number of epochs and best for a high number of epochs.* **The REX profile performs best for a small and medium number of epochs.** While the Step schedule is consistently used to achieve state-of-the-art results in Computer Vision (He et al., 2016; Zagoruyko & Komodakis, 2016; Huang et al.,

Table 4. RN20-CIFAR10. The number of epochs was predefined before the execution of the algorithms. **Bold red** indicates Top-1 performance, **black bold** is Top-3.

SGDM	1%	5%	10%	25%	50%	100%
+ Step Schedule	32.14 ± .34	14.94 ± .27	11.80 ± .11	**8.82** ± .25	8.43 ± .07	**7.32** ± .14
+ Cosine Schedule	**28.49** ± .25	**13.05** ± .17	**10.62** ± .29	**8.80** ± .08	**8.10** ± .13	7.78 ± .14
+ OneCycle	40.14 ± 2.62	18.93 ± 1.85	12.74 ± .36	10.83 ± .25	9.23 ± .19	8.42 ± .12
+ Linear Schedule	**28.70** ± 1.13	**13.09** ± .13	**10.85** ± .15	9.03 ± .24	**8.15** ± .12	**7.62** ± .12
+ Decay on Plateau	41.98 ± 3.20	25.93 ± .45	11.29 ± .35	9.05 ± .07	8.26 ± .07	7.97 ± .14
+ Exp decay	31.31 ± 1.34	14.85 ± .38	11.56 ± .22	9.55 ± .09	9.20 ± .13	7.82 ± .05
+ REX	**27.94** ± .46	**12.86** ± .27	**10.23** ± .13	**8.37** ± .09	**7.52** ± .24	**7.52** ± .05
Adam	42.10 ± 2.71	23.01 ± 1.10	16.58 ± .18	13.63 ± .22	11.90 ± .06	11.94 ± .06
+ Step Schedule	30.72 ± .16	15.41 ± .26	12.20 ± .11	10.47 ± .10	**8.75** ± .17	**8.55** ± .05
+ Cosine Schedule	**29.20** ± .24	**14.31** ± .28	**11.45** ± .27	**9.56** ± .12	9.15 ± .12	8.93 ± .07
+ OneCycle	37.17 ± 2.49	16.16 ± .19	14.11 ± .57	10.33 ± .20	9.87 ± .12	9.03 ± .18
+ Linear Schedule	**28.99** ± .37	**14.08** ± .34	**10.97** ± .19	**9.25** ± .12	9.20 ± .22	8.89 ± .05
+ Decay on Plateau	43.40 ± 4.57	22.21 ± .96	13.46 ± .38	9.71 ± .39	**8.92** ± .18	8.80 ± .11
+ Exp decay	31.87 ± .59	15.82 ± .06	12.91 ± .21	10.48 ± .15	9.24 ± .16	**8.53** ± .07
+ REX	**27.64** ± .02	**13.96** ± .16	**10.88** ± .05	**9.44** ± .22	**8.72** ± .24	**8.18** ± .15

2017; Hu et al., 2017; Redmon et al., 2016; He et al., 2018), it does not translate directly to lower epoch settings.

A new profile. Since there is no profile which performs optimally across sampling rates, it remains to ask if there is a profile and sampling rate combination that results in strong performance in both low and high epoch settings. Therefore, we propose the Reflected Exponential (REX) profile; see Figure 1. REX is an alternative to the linear and exponential profile, and we find that REX has stronger empirical performance in the budgeted setting. REX performs best with a per-iteration sampling rate, similar to the linear schedule. We evaluate the performance of REX extensively in following sections.

We also motivate REX with the empirical observation that the linear schedule can be improved in some cases by delaying the onset of the decay, i.e., holding the initial learning rate constant until XX% of the budget, and then linearly decaying the learning rate to 0; see Figure 2. In particular, it appears that performance can be improved with such delay in the high epoch regime, but this strategy is less effective with fewer epochs. However, *the exact onset of the delay introduces an additional hyperparameter*. REX can be understood as an interpolation between a linear schedule and a delayed linear schedule without additional hyperparameters. Furthermore, REX generally outperforms the linear schedule, which has been previously suggested as the best budgeted schedule (Li et al., 2020), for small and large epochs.

It appears that certain schedules have reasonable performance across sampling rates, while others have poor or state-of-the-art performance depending on the sampling rate. If the sampling rate is unknown or there is a particular reason to select a low sampling rate, the approximated step profile appears to be the best choice. However, in most applications, the sampling rate is a choice by the practitioner. Since the REX profile with a per-iteration sampling rate generally performs the best, there may be limited motivation to use alternative schedules.

4 Results

In this section we present results in all seven experimental settings given in Table 3, including image classification, image generation, object detection and natural language processing. For fair evaluation in the budgeted training scenario, only the learning rate is tuned in multiples of 3 for each schedule, setting, and number of epochs. All reported metrics are averaged across three separate trials. We run all settings at 1%, 5%, 10%, 25%, 50%, and 100% of maximum epochs, representing both low and high budgets. In each setting, the learning rate schedule is concerned only with the total epochs for that run, e.g., the linear schedule will decay linearly to 0 regardless if the budget is 1% or 100% of the maximum epochs. For BERT_{BASE}-GLUE, results are given for 1 run and at $\frac{1}{3}$, $\frac{2}{3}$, and $\frac{3}{3}$ of total epochs. The maximum total epochs is determined from commonly used epochs in the literature, and validated to achieve the reported score in the literature. The maximum epochs is given in Table 3. The goal is to demonstrate performance in both the low and high budget regime across a range of common applications to instill confidence that the proposed schedule

Table 5. WRN-STL10. The number of epochs was predefined before the execution of the algorithms. **Bold red** indicates Top-1 performance, **black bold** is Top-3.

SGDM	1%	5%	10%	25%	50%	100%
+ Step Schedule	60.09 ± 1.15	38.12 ± .32	33.86 ± .10	22.42 ± .56	**17.20** ± .35	**14.51** ± .26
+ Cosine Schedule	**57.81** ± 1.05	37.42 ± .29	**27.51** ± .25	**20.03** ± .26	**17.02** ± .24	14.66 ± .25
+ OneCycle	58.75 ± .76	**36.90** ± .37	**26.97** ± .27	21.67 ± .27	19.69 ± .21	19.00 ± .42
+ Linear Schedule	**58.74** ± 1.26	**34.81** ± .40	28.17 ± .64	**19.54** ± .20	17.39 ± .24	**14.58** ± .18
+ Decay on Plateau	59.64 ± .92	37.64 ± 1.44	36.94 ± 1.96	21.05 ± .27	17.83 ± .39	15.16 ± .36
+ Exp decay	60.21 ± .77	38.94 ± 1.08	34.11 ± .77	22.65 ± .49	20.60 ± .21	15.85 ± .28
+ REX	**55.93** ± .46	**34.50** ± .16	**25.52** ± .17	**20.54** ± .32	**16.97** ± .46	**14.60** ± .31
Adam	58.65 ± 1.79	42.66 ± .68	33.17 ± 1.94	23.35 ± .20	**19.63** ± .26	18.65 ± .07
+ Step Schedule	59.35 ± .98	47.14 ± .42	35.10 ± 1.10	23.85 ± .07	**19.63** ± .33	**18.29** ± .10
+ Cosine Schedule	58.95 ± .95	40.69 ± 1.09	**31.00** ± .74	22.85 ± .47	21.47 ± .31	19.08 ± .36
+ OneCycle	**57.88** ± .88	**36.41** ± .29	**27.90** ± .63	**20.02** ± .19	**19.21** ± .28	19.03 ± .43
+ Linear Schedule	**56.72** ± .22	**40.25** ± 1.00	31.15 ± .29	**21.70** ± .11	21.53 ± .44	**17.85** ± .15
+ Decay on Plateau	58.72 ± .60	42.30 ± .68	33.00 ± .80	22.77 ± .33	19.91 ± .45	19.61 ± .56
+ Exp decay	58.92 ± .52	44.76 ± .90	33.52 ± 1.18	23.30 ± .39	20.70 ± .50	19.63 ± .24
+ REX	**56.47** ± .31	**35.52** ± .44	**27.24** ± .20	**21.65** ± .21	**19.12** ± .31	**17.75** ± .22

will work "in the wild". We use a model-dataset-optimizer notation, e.g. RN20-CIFAR10-SGDM means a ResNet20 model trained on CIFAR10 with momentum SGD.

4.1 Learning Rate Schedules

There are many popular learning rate schedules implemented in widely-used frameworks and packages. In general, the schedules are aware of the current time step t and the maximum time step T. Let η denote the learning rate and β the momentum. We comprehensively detail the schedules considered in this paper below, covering almost all widely-implemented schedules; see Figure 1 for a visualization.

- Step schedule (He et al., 2016): $\eta_t = \gamma_t \cdot \eta_0$ where γ_t is piece-wise and depends on $\frac{t}{T}$. A typical schedule (He et al., 2016) would be to decay the learning rate by 0.1 at $\frac{1}{2}$ epochs and again by 0.1 at $\frac{3}{4}$ epochs. We employ such a step schedule for all our experiments.
- Decay on Plateau (Abadi et al., 2015; Paszke et al., 2017): A practical version of the step schedule, where the learning rate is decayed when the validation loss does not improve for certain number of tuneable epochs, which we tune in multiples of 5.
- Linear schedule (Abadi et al., 2015; Paszke et al., 2017): $\eta_t = \left(1 - \frac{t}{T}\right) \cdot \eta_0$.
- Cosine schedule (Loshchilov & Hutter, 2017b): $\eta_t = \frac{\eta_0}{2} \cdot \left(1 + \cos(\frac{\pi \cdot t}{T})\right)$.
- Exponential schedule (Abadi et al., 2015; Paszke et al., 2017): $\eta_t = \eta_0 \cdot e^{\frac{\gamma t}{T}}$. We find that setting $\gamma = -3$ yields the best performance.
- OneCycle schedule (Smith, 2018):

$$\eta_t = \begin{cases} \eta_{\min} + (\eta_{\max} - \eta_{\min}) \left(\frac{t}{\frac{T}{2}}\right) \eta_0, \textbf{if } \frac{t}{T} < \frac{1}{2} \\ \eta_{\min} + (\eta_{\max} - \eta_{\min}) \left(2 - \frac{t}{\frac{T}{2}}\right) \eta_0, \text{else} \end{cases}$$

$$\beta_t = \begin{cases} \beta_{\min} + (\beta_{\max} - \beta_{\min}) \left(1 - \frac{t}{\frac{T}{2}}\right) \beta_0, \textbf{if } \frac{t}{T} < \frac{1}{2} \\ \beta_{\min} + (\beta_{\max} - \beta_{\min}) \left(\frac{t}{\frac{T}{2}} - 1\right) \beta_0, \text{else} \end{cases}$$

  $\eta_{\min}$, $\eta_{\max}$, $\beta_{\min}$, and $\beta_{\max}$ are hyperparameters. For fair computational comparison, we follow the recommended settings (Smith, 2018) and set $\eta_{\min} = \eta_{\max} \cdot 0.1$, $\beta_{\max} = 0.95$, $\beta_{\min} = 0.85$, so that $\eta_{\max}$ is the only hyperparameter.
- REX schedule:

$$\eta_t = \eta_0 \cdot \left(\frac{1 - \frac{t}{T}}{\frac{1}{2} + \frac{1}{2} \cdot (1 - \frac{t}{T})}\right).$$

  We re-emphasize the motivation for REX: it is a new profile and sampling rate combination, which is motivated by the improved performance of a delayed linear schedule in certain circumstances. REX aggressively decreases the learning rate towards the end of the training process, which is the "reflection" of the exponential decay.

There are simply too many schedules to compare comprehensively, so we select the widely-used schedules above for comparison. We apply the schedules to the two most popular optimizers: SGD with momentum and Adam.

Table 6. `VGG16-CIFAR100` generalization error. The number of epochs was predefined before the execution of the algorithms. **Bold red** indicates Top-1 performance, **black bold** is Top-3.

SGDM	1%	5%	10%	25%	50%	100%
+ Step Schedule	95.03 ± .42	69.87 ± .28	46.97 ± .13	35.04 ± .24	30.09 ± .32	**27.83** ± .30
+ Cosine Schedule	95.03 ± .42	61.82 ± .13	**41.26** ± .26	**31.93** ± .09	**28.63** ± .11	**27.84** ± .12
+ OneCycle	**91.96** ± 1.01	**58.35** ± .40	45.39 ± .73	32.62 ± .21	30.10 ± .34	29.09 ± .12
+ Linear Schedule	96.11 ± 1.64	**58.14** ± 1.19	**39.66** ± .61	**31.95** ± .29	**29.10** ± .34	28.26 ± .08
+ Decay on Plateau	**94.70** ± 1.20	65.25 ± 1.72	50.81 ± .58	35.29 ± .59	30.65 ± .31	29.74 ± .43
+ Exp decay	96.54 ± .39	65.65 ± 1.24	49.04 ± 1.98	33.15 ± .19	29.51 ± .22	28.47 ± .18
+ REX	**94.92** ± .91	**56.62** ± .65	**40.72** ± .29	**31.16** ± .11	**28.54** ± .02	**27.27** ± .30
Adam	92.70 ± .50	64.05 ± .41	57.56 ± 1.30	37.98 ± .20	33.62 ± .11	31.09 ± .09
+ Step Schedule	92.65 ± .38	62.90 ± .08	44.94 ± .49	34.16 ± .11	29.40 ± .22	**27.75** ± .15
+ Cosine Schedule	**91.48** ± .42	**55.90** ± 2.46	**40.31** ± .07	**32.32** ± .14	29.68 ± .17	**28.08** ± .10
+ OneCycle	**92.18** ± .69	58.29 ± .53	43.47 ± .28	34.59 ± .31	29.83 ± .29	29.58 ± .18
+ Linear Schedule	92.94 ± .49	**54.32** ± 1.17	**39.49** ± .11	**32.01** ± .49	**29.30** ± .18	28.65 ± .10
+ Decay on Plateau	92.76 ± .48	64.10 ± .22	57.05 ± .84	32.60 ± .31	**29.03** ± .10	28.67 ± .19
+ Exp decay	92.43 ± .67	55.26 ± 1.24	42.62 ± .12	32.37 ± .18	29.53 ± .12	28.83 ± .08
+ REX	**91.93** ± .01	**52.20** ± .47	**39.51** ± .21	**31.68** ± .57	**28.58** ± .16	**26.99** ± .09

Table 7. `VAE-MNIST` generalization loss. The number of epochs was predefined before the execution of the algorithms. **Bold red** indicates Top-1 performance, **black bold** is Top-3, ignoring non SGDM and Adam optimizers.

SGDM	1%	5%	10%	25%	50%	100%
+ Step Schedule	180.30 ± 6.98	152.97 ± .55	146.24 ± 2.50	140.28 ± .51	137.70 ± .93	136.34 ± .31
+ Cosine Schedule	174.52 ± 1.09	**145.99** ± .15	**141.23** ± .36	**139.15** ± .26	**136.69** ± .27	**135.05** ± .09
+ OneCycle	**161.95** ± .67	146.25 ± .35	**143.01** ± 1.08	**139.79** ± .66	**137.20** ± .06	**135.65** ± .44
+ Linear Schedule	174.64 ± .15	**146.15** ± .26	143.64 ± .80	148.00 ± .48	141.72 ± .48	137.84 ± .32
+ Decay on Plateau	**167.16** ± .30	151.15 ± .11	146.82 ± .58	140.51 ± .73	139.54 ± .34	137.33 ± .49
+ Exp decay	179.60 ± 3.47	160.52 ± .64	146.24 ± .73	154.31 ± .43	145.83 ± .48	139.67 ± .57
+ REX	**149.85** ± 1.62	**139.56** ± .78	**137.15** ± .05	**134.41** ± .78	**135.69** ± .24	**135.03** ± .37
Adam	152.10 ± .55	142.54 ± .50	140.10 ± .82	136.28 ± .18	134.64 ± .14	134.66 ± .17
+ Step Schedule	153.45 ± 1.47	142.19 ± .98	138.32 ± .20	136.62 ± .30	134.14 ± .56	133.34 ± .41
+ Cosine Schedule	149.82 ± .32	140.78 ± .72	**137.66** ± .79	134.73 ± .04	**133.25** ± .26	133.23 ± .30
+ OneCycle	**149.07** ± .99	**139.75** ± .27	138.12 ± .99	**134.67** ± .55	**133.27** ± .07	**132.83** ± .33
+ Linear Schedule	**148.93** ± .20	**139.82** ± .20	**137.00** ± .70	**134.71** ± .25	134.00 ± .49	**132.95** ± .24
+ Decay on Plateau	152.08 ± .45	141.54 ± .31	139.76 ± .52	135.68 ± .59	134.10 ± .21	134.06 ± .45
+ Exp decay	149.28 ± .46	142.94 ± 1.28	138.82 ± .36	135.19 ± .43	134.05 ± .16	133.88 ± .85
+ REX	**148.59** ± .33	**139.05** ± .20	**136.62** ± .21	**134.24** ± .02	**133.16** ± .05	**132.52** ± .05

4.2 Empirical Results

Image Classification. We choose four diverse settings for this task. For datasets, we use the standard CIFAR10 and CIFAR100 datasets, in addition to the low count, high-res STL10 dataset, as well as the standard ImageNet dataset. Since ResNets remain the most commonly-deployed model in industry, we perform experiments with three variations of the ResNet (He et al., 2016). The ResNet20 comes from the line of lower cost, lower performance ResNets, and is a close cousin of the more expensive and better performing ResNet18. ResNet50 belongs to the latter series, and is a standard model for ImageNet. We also include the Wide ResNet variation which further increases the model width for better performance (Zagoruyko & Komodakis, 2016). The other model we employ is the VGG-16 model (Simonyan & Zisserman, 2014). While VGG models are far outdated in attaining state-of-the-art performance, the archi-

Table 8. YOLO-VOC mAP. The number of epochs was predefined before the execution of the algorithms. **Bold red** indicates Top-1 performance, **black bold** is Top-3.

	1%	5%	10%	25%	50%	100%
Adam	45.0 ± 3.4	48.1 ± 7.6	61.9 ± 1.8	70.2 ± 3.5	72.1 ± 6.4	79.1 ± 1.6
+ Step Schedule	62.2 ± 1.7	**67.0** ± 3.4	71.8 ± 1.0	78.5 ± 0.2	81.1 ± 1.0	83.2 ± 0.2
+ OneCycle	60.4 ± 7.2	63.8 ± 7.6	74.9 ± 1.0	79.9 ± 1.3	81.1 ± 2.8	83.3 ± 0.4
+ Cosine Schedule	**63.6** ± 5.2	66.8 ± 6.1	**75.9** ± 0.2	**81.1** ± 0.7	**82.5** ± 1.0	**84.0** ± 0.2
+ Linear Schedule	**63.7** ± 5.5	**67.2** ± 5.9	**76.2** ± 0.7	**81.1** ± 0.9	**82.4** ± 1.2	**83.4** ± 0.2
+ Exp decay	49.6 ± 24	**68.1** ± 4.6	75.6 ± 0.1	80.1 ± 0.7	81.2 ± 2.2	83.2 ± 0.2
+ REX	**64.0** ± 5.0	**67.0** ± 6.5	**76.7** ± 0.3	**81.2** ± 0.7	**82.2** ± 1.8	**83.4** ± 0.4

tecture is still relevant for custom applications with smaller CNNs, where residual connections have limited application. We provide thorough evaluation in the RN20-CIFAR10, WRN-STL10, VGG16-CIFAR100 settings, and, due to computational constraints, provide lower epochs results for RN50-ImageNet, given in Tables 4, 5, 6, and 9.

As observed in (Li et al., 2020), the linear schedule performs well for both SGD and Adam, particularly for a low number of epochs. While the Step schedule performs well for the maximum number of epochs, it scales very poorly to lower epoch settings. On the other hand, REX performs well in both high and low epoch regimes. Results also follow general Computer Vision observations for these settings, where SGD tends to outperform Adam.

Table 9. RN50-ImageNet generalization error. The number of epochs was predefined before the execution of the algorithms. **Bold red** indicates Top-1 performance, **black bold** is Top-3.

SGDM	1%	5%
+ Step Schedule	87.28	46.58
+ Cosine Schedule	**82.88**	**43.90**
+ OneCycle	90.94	55.00
+ Linear Schedule	**82.00**	**43.27**
+ Exp decay	90.19	48.28
+ REX	**80.98**	**40.78**
Adam	**1%**	**5%**
+ Step Schedule	77.97	45.91
+ Cosine Schedule	**73.51**	**43.66**
+ OneCycle	82.58	62.57
+ Linear Schedule	**71.42**	**42.01**
+ Exp decay	75.54	45.43
+ REX	**69.91**	**40.65**

Image Generation. The two most popular types of networks for image generation are Variational Encoders (VAE) (Kingma & Welling, 2015) and Generative Adversarial Networks (GAN) (Goodfellow et al., 2014). However, out of the two, only VAEs consistently benefit from learning rate decay (Goodfellow et al., 2014; Chen et al., 2016; Arjovsky et al., 2017; Brock et al., 2019; Hou et al., 2016; Sonderby et al., 2016; Vahdat & Kautz, 2021). Therefore, we select VAEs as the network of choice for image generation. We train VAEs on the MNIST dataset for 200 epochs, after which performance no longer improves. Results are given in Table 7.

Table 10. Results of BERT$_{BASE}$-GLUE. AdamW + Linear Schedule follows the huggingface (et al., 2020b) implementation, and achieves the results in well-known studies (Devlin et al., 2019; Sanh et al., 2020). Results given by 1 epoch/2 epochs/3 epochs. Excluding the problematic WNLI dataset (Devlin et al., 2019).

	Score
AdamW	79.9/81.2/81.8
+ Step Schedule	80.2/81.9/82.3
+ Cosine Schedule	80.9/**82.2**/**82.7**
+ OneCycle	**81.0**/82.0/**82.7**
+ Linear Schedule	**81.2**/**82.3**/82.6
+ Exp decay	80.6/81.8/82.5
+ REX	**81.7**/**82.6**/**82.8**

The linear schedule performs well for Adam, but not for SGDM. Similarly, the cosine schedule performs well for SGDM, but not for Adam. The OneCycle schedule performs well across all settings, but REX outperforms all other schedules in the low budget and high budget setting.

Object Detection. We train a YOLOv3 (Redmon & Farhadi, 2018) model on the Pascal VOC dataset. The training set is the combined 2007 and 2012 training set, and the test set is the 2007 test set. We were able to achieve the mAP score reported in the literature by training the network for 50 epochs. Thus, we set this as the maximum number of epochs. We find that the network does not train well without a warm-up period, so all networks are trained for 2 epochs

Table 11. Results of BERT$_{BASE}$-GLUE. AdamW + Linear Schedule follows the huggingface (et al., 2020b) implementation, and achieves the results in well-known studies (Devlin et al., 2019; Sanh et al., 2020). Results given by 1 epoch/2 epochs/3 epochs. Excluding the problematic WNLI dataset (Devlin et al., 2019).

	CoLA	MNLI	MRPC	QNLI	QQP	RTE	SST-2	STS-B
AdamW	54.8/54.7/55.2	82.9/83.3/83.7	84.8/87.2/87.6	88.7/**90.4**/**90.7**	89.4/90.2/90.5	59.2/64.6/66.8	91.2/91.3/91.2	**87.8**/87.8/88.3
+ Step Schedule	53.5/56.9/56.6	82.6/83.4/83.9	85.6/87.9/88.3	88.2/90.1/**90.4**	89.0/90.5/90.6	**63.5/65.7**/**67.5**	**92.8**/92.8/**93.0**	86.7/88.0/88.4
+ Cosine Schedule	55.7/**58.6**/58.2	**83.5/84.0/84.2**	84.5/87.6/87.9	**89.4**/89.8/**90.4**	**89.8/90.6**/**91.0**	**64.2**/65.3/**67.5**	**92.7**/**93.1**/**93.7**	87.4/88.4/**88.7**
+ OneCycle	**57.7/58.1**/56.5	**83.6**/83.8/**84.2**	**87.3**/87.5/**89.9**	**89.5/91.0/90.7**	**89.8/90.6**/90.8	60.3/63.9/**67.5**	92.1/92.2/**93.0**	**88.1/88.5/89.0**
+ Linear Schedule	**58.0**/57.6/**58.8**	**83.5**/**84.1**/**84.3**	85.4/**88.1**/88.0	88.8/**90.4**/89.6	89.7/**90.6**/**91.0**	**63.5/65.7**/67.1	**92.8**/**93.0**/92.9	**87.9/88.5/88.8**
+ Exp decay	57.5/57.3/**59.1**	**83.6**/83.9/84.1	**86.2/88.7/89.1**	88.2/89.2/89.6	88.8/90.3/90.6	61.0/63.9/66.0	92.1/**93.1**/**93.0**	87.2/88.2/88.5
+ REX	**57.8**/**58.8**/**59.1**	83.4/**84.0**/**84.3**	**87.3**/**88.9**/**89.1**	**88.9/90.5**/90.3	**90.0**/**90.7**/**91.0**	**65.3**/**66.8**/67.1	**92.7**/92.7/92.7	87.6/**88.6**/88.6

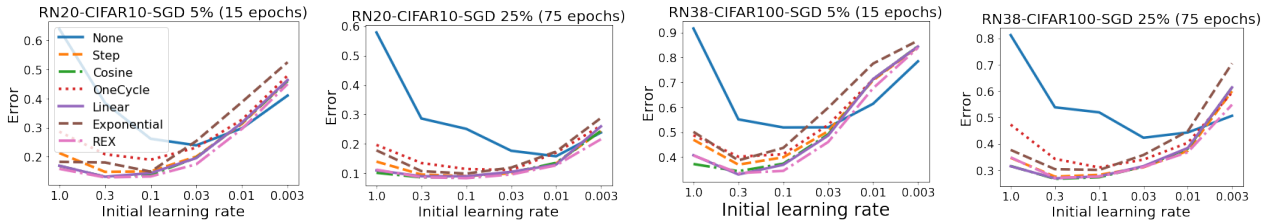

Figure 3. Error against initial learning for RN20-CIFAR10-SGD and RN38-CIFAR100-SGD for 5% and 25% of total epochs. As expected all, schedules suffer as the learning rate grows too large or too small.

from a learning rate of 1e-5 linearly increased to 1e-4. This warm-up phase is not counted as part of the allocated training budget. We also round up the number of epochs to the closest integer: for example, the 1% setting trains for 2 warmup epochs and then $\lceil 50 \cdot 0.01 \rceil = 1$ epoch, for a total of 3 epochs. The 100% setting trains for 2 warmup epochs and then 50 epochs for a total of 52 epochs. Results are given in Table 8. Similar to other settings, the step schedule performs reasonably well for a large number of epochs, but is outperformed by the cosine schedule. REX performs well in the low epoch setting.

Natural Language Processing. Fine-tuning pre-trained transformer models is one of the most common training procedures in NLP (Devlin et al., 2019; et al., 2020a), thus making it a setting of interest. This is because i) it is often cost-prohibitive for practitioners to pre-train their own models and ii) fine-tuning pre-trained transformers often results in significantly better performance in comparison to training a smaller model from scratch. The linear schedule is the default schedule implemented in HuggingFace (et al., 2020b), the most popular package for transformer models, and is considered the gold standard in this domain. We fine-tune BERT$_{BASE}$ on the GLUE benchmark, an NLP benchmark with nine datasets. We leave out the problematic WNLI dataset (Devlin et al., 2019). Since we are able to attain the scores reported in the literature with 3 epochs of fine-tuning, we set that as the maximum number of epochs. Due to computational constraints, we can only perform one run per setting, which causes some variability within the results. Although REX achieves the best mean score for small and large budgets, we see that the best optimizer can vary depending on the dataset. For example, OneCycle attains the best scores on QNLI and MRPC, and the Cosine schedule performs the best on SST-2.

Sensitivity to learning rate tuning. While it is reasonable to suggest that the practitioner simply pick a per-iteration sampling rate for the REX, linear, and other profiles, a relevant issue in budgeted training is performance given a limited number of experimental trials. Namely, in extreme cases, the practitioner may not even have the budget to finely tune the learning rate. Therefore, we plot the considered schedules in two settings against learning rate, presented in Figure 3. Clearly, there is no schedule that can recover from a poor initial learning rate. However, schedules tend to retain their relative ordering across initial learning rates. This means that even with poor hyperparameter settings, the choice of learning rate schedule remains important. REX, represented by the pink line below all other lines, outperforms other schedules for most learning rates in the budgeted settings presented in the plots.

5 Conclusion

In this paper, we identified issues with existing learning rate schedules in the budgeted setting. We proposed a profile and sampling rate framework for understanding existing schedules. While there is no optimal profile, we found that the proposed REX schedule performs well with a sampling rate of every iteration in both small and large epoch regimes. With thorough empirical evaluation, we confirm that the

proposed REX learning rate schedule performs favorably across a large number of settings including image classification, generation, object detection, and natural language processing.

A EXPERIMENTS

We describe the nine test problems in this paper.

- **CIFAR10 - ResNet20.** CIFAR10 contains 60,000 32x32x3 images with a 50,000 training set, 10,000 test set split. There are 10 classes. ResNet20 (He et al., 2016) is an 20 layers deep CNN with skip connections for image classification. Trained with a batch size of 128.
- **TINY IMAGENET - ResNet56.** Tiny ImageNet contains 110,000 64x64x3 images with a 100,000 training set, 10,000 test set split. There are 200 classes. ResNet56 (He et al., 2016) is a 56 layer deep CNN with skip connections for image classification. Trained with a batch size of 128.
- **CIFAR100 - VGG16.** CIFAR100 is a fine-grained version of CIFAR-10 and contains 60,000 32x32x3 images with a 50,000 training set, 10,000 test set split. There are 100 classes. VGG16 (Simonyan & Zisserman, 2014) is a 16 layers deep CNN with extensive use of 3x3 convolutional filters. Trained with a batch size of 128.
- **STL10 - Wide ResNet 16-8.** STL10 contains 1300 96x96x3 images with a 500 training set, 800 test set split. There are 10 classes. Wide ResNet 16-8 (Zagoruyko & Komodakis, 2016) is a 16 layers deep ResNet which is 8 times wider. Trained with a batch size of 64.
- **PTB - LSTM.** PTB is an English text corpus containing 929,000 training words, 73,000 validation words, and 82,000 test words. There are 10,000 words in the vocabulary. The model is stacked LSTMs (Hochreiter & Schmidhuber, 1997) with 2 layers, 650 units per layer, and dropout of 0.5. Trained with a batch size of 20. We use the official TensorFlow v1 implementation for PTB - LSTM.
- **FMNIST - CAPS.** FMNIST contains 60,000 32x32x1 grayscale images with a 50,000 training set, 10,000 test set split. There are 10 classes of 10 clothing items. Capsule Networks (Sabour et al., 2017) represent Neural Networks as a set of capsules, where each capsule encodes a specific entity or meaning. The activations of capsules depend on comparing incoming pose predictions, as opposed to standard neural networks. The Capsule Network uses 3 iterations in the routing algorithm. Trained with a batch size of 128.
- **MNIST - VAE.** MNIST contains 60,000 32x32x1 grayscale images with a 50,000 training set, 10,000 test set split. There are 10 classes of 10 digits. VAE (Kingma & Welling, 2015) with three dense encoding layers and three dense decoding layers with a latent space of size 2. Trained with a batch size of 100.
- **CIFAR10 - NCSN.** CIFAR10 contains 60,000 32x32x3 images with a 50,000 training set, 10,000 test set split. There are 10 classes. NCSN (Song & Ermon, 2019) is a recent state-of-the-art generative model which achieves the best reported inception score. We compute inception scores based on a total of 50000 samples. Since DEMON depends on a predefined number of epochs, we evaluate inception score at the end of training; otherwise, we follow the exact implementation in and defer details to the original paper.
- **GLUE - BERT.** The GLUE benchmark (Wang et al., 2019) consists of 9 different language tasks (Warstadt et al., 2018; Socher et al., 2013; Dolan & Brockett, 2005; Agirre et al., 2007; Williams et al., 2018; Rajpurkar et al., 2016; Dagan et al., 2006; Bar Haim et al., 2006; Giampiccolo et al., 2007; Bentivogli et al., 2009; Levesque et al., 2011), grouped together to form a benchmark. BERT (Devlin et al., 2019) is a relatively recently proposed language model which has become the standard for many tasks in NLP. In particular, BERT can be fine-tuned to an array of tasks, and here we evaluate the fine-tuning procedure of BERT to the GLUE benchmark.

REFERENCES

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL `http://tensorflow.org/`. Software available from tensorflow.org.

Agirre, E., M'arquez, L., and Wicentowski, R. (eds.). *Proceedings of the Fourth International Workshop on Semantic Evaluations (SemEval-2007)*. Association for Computational Linguistics, Prague, Czech Republic, June 2007.

Arjovsky, M., Chintala, S., and Bottou, L. Wasserstein gan, 2017.

Bar Haim, R., Dagan, I., Dolan, B., Ferro, L., Giampiccolo, D., Magnini, B., and Szpektor, I. The second PASCAL recognising textual entailment challenge. 2006.

Bentivogli, L., Dagan, I., Dang, H. T., Giampiccolo, D., and Magnini, B. The fifth PASCAL recognizing textual entailment challenge. 2009.

Brock, A., Donahue, J., and Simonyan, K. Large scale gan training for high fidelity natural image synthesis, 2019.

Chang, B., Meng, L., Haber, E., Ruthotto, L., Begert, D., and Holtham, E. Reversible architectures for arbitrarily deep residual neural networks, 2017.

Chen, J., Wolfe, C., Li, Z., and Kyrillidis, A. Demon: Momentum decay for improved neural network training, 2020a.

Chen, T., Kornblith, S., Swersky, K., Norouzi, M., and Hinton, G. Big self-supervised models are strong semi-supervised learners, 2020b.

Chen, X., Duan, Y., Houthooft, R., Schulman, J., Sutskever, I., and Abbeel, P. Infogan: Interpretable representation learning by information maximizing generative adversarial nets, 2016.

Dagan, I., Glickman, O., and Magnini, B. The PASCAL recognising textual entailment challenge. In *Machine learning challenges. evaluating predictive uncertainty, visual object classification, and recognising tectual entailment*, pp. 177–190. Springer, 2006.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019.

Dolan, W. B. and Brockett, C. Automatically constructing a corpus of sentential paraphrases. In *Proceedings of the International Workshop on Paraphrasing*, 2005.

Duchi, J., Hazan, E., and Singer, Y. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.

et al., T. B. B. Language models are few-shot learners, 2020a.

et al., T. W. Huggingface's transformers: State-of-the-art natural language processing, 2020b.

Giampiccolo, D., Magnini, B., Dagan, I., and Dolan, B. The third PASCAL recognizing textual entailment challenge. In *Proceedings of the ACL-PASCAL workshop on textual entailment and paraphrasing*, pp. 1–9. Association for Computational Linguistics, 2007.

Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. Generative adversarial networks, 2014.

Goyal, P., Dollr, P., Girshick, R., Noordhuis, P., Wesolowski, L., Kyrola, A., Tulloch, A., Jia, Y., and He, K. Accurate, large minibatch sgd: Training imagenet in 1 hour, 2018.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on CVPR*, pp. 770–778, 2016.

He, K., Gkioxari, G., Dollr, P., and Girshick, R. Mask r-cnn, 2018.

Hinton, G., Srivastava, N., and Swersky, K. Neural networks for machine learning lecture 6a overview of mini-batch gradient descent. *Cited on*, 14:8, 2012.

Hochreiter, S. and Schmidhuber, J. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

Hou, X., Shen, L., Sun, K., and Qiu, G. Deep feature consistent variational autoencoder, 2016.

Hu, J., Shen, L., Albanie, S., Sun, G., and Wu, E. Squeeze-and-excitation networks. *arxiv preprint arXiv:1709.01507*, 2017.

Huang, G., Liu, Z., Van Der Maaten, L., and Weinberger, K. Q. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4700–4708, 2017.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Kingma, D. P. and Welling, M. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2015.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. Imagenet classification with deep convolutional neural networks. In *Advances in NeurIPS*, pp. 1097–1105, 2012.

Levesque, H. J., Davis, E., and Morgenstern, L. The Winograd schema challenge. In *AAAI Spring Symposium: Logical Formalizations of Commonsense Reasoning*, volume 46, pp. 47, 2011.

Li, M., Yumer, E., and Ramanan, D. Budgeted training: Rethinking deep neural network training under resource constraints, 2020.

Li, Z. and Arora, S. An exponential learning rate schedule for deep learning. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=rJg8TeSFDH.

Liu, L., Jiang, H., He, P., Chen, W., Liu, X., Gao, J., and Han, J. On the variance of the adaptive learning rate and beyond, 2020.

Loshchilov, I. and Hutter, F. Fixing weight decay regularization in adam. *arXiv preprint arXiv:1711.05101*, 2017a.

Loshchilov, I. and Hutter, F. Sgdr: Stochastic gradient descent with warm restarts, 2017b.

Lucas, J., Sun, S., Zemel, R., and Grosse, R. Aggregated momentum: Stability through passive damping. *arXiv preprint arXiv:1804.00325*, 2018.

Odonoghue, B. and Candes, E. Adaptive restart for accelerated gradient schemes. *Foundations of computational mathematics*, 15(3):715–732, 2015.

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., and Lerer, A. Automatic differentiation in pytorch. 2017.

Rajpurkar, P., Zhang, J., Lopyrev, K., and Liang, P. Squad: 100,000+ questions for machine comprehension of text. In *Proceedings of EMNLP*, pp. 2383–2392. Association for Computational Linguistics, 2016.

Redmon, J. and Farhadi, A. Yolov3: An incremental improvement, 2018.

Redmon, J., Divvala, S., Girshick, R., and Farhadi, A. You only look once: Unified, real-time object detection, 2016.

Sabour, S., Fross, N., and Hinton, G. Dynamic routing between capsules. In *Advances in neural information processing systems*, 2017.

Sanh, V., Debut, L., Chaumond, J., and Wolf, T. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter, 2020.

Shawahna, A., Sait, S. M., and El-Maleh, A. Fpga-based accelerators of deep learning networks for learning and classification: A review. *IEEE Access*, 7:78237859, 2019. ISSN 2169-3536. doi: 10.1109/access.2018.2890150. URL http://dx.doi.org/10.1109/ACCESS.2018.2890150.

Simonyan, K. and Zisserman, A. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Smith, L. A disciplined approach to neural network hyperparameters: Part 1 – learning rate, batch size, momentum, and weight decay. *arXiv preprint arXiv:1803.09820*, 2018.

Smith, L. N. Cyclical learning rates for training neural networks, 2017.

Smith, S., Kindermans, P.-J., Ying, C., and Le, Q. Don't decay the learning rate, increase the batch size. *arXiv preprint arXiv:1711.00489*, 2017.

Socher, R., Perelygin, A., Wu, J., Chuang, J., Manning, C. D., Ng, A., and Potts, C. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of EMNLP*, pp. 1631–1642, 2013.

Sonderby, C. K., Raiko, T., Maaloe, L., Sonderby, S. K., and Winther, O. Ladder variational autoencoders, 2016.

Song, Y. and Ermon, S. Generative modeling by estimating gradients of the data distribution. *arXiv preprint arXiv:1907.05600*, 2019.

Sutskever, I., Martens, J., Dahl, G., and Hinton, G. On the importance of initialization and momentum in deep learning. In *International conference on machine learning*, pp. 1139–1147, 2013.

Sze, V., Chen, Y.-H., Emer, J., Suleiman, A., and Zhang, Z. Hardware for machine learning: Challenges and opportunities. *2017 IEEE Custom Integrated Circuits Conference (CICC)*, Apr 2017. doi: 10.1109/cicc.2017.7993626. URL http://dx.doi.org/10.1109/CICC.2017.7993626.

Tripathi, S., Lipton, Z. C., Belongie, S., and Nguyen, T. Context matters: Refining object detection in video with recurrent neural networks, 2016.

Vahdat, A. and Kautz, J. Nvae: A deep hierarchical variational autoencoder, 2021.

Wang, A., Singh, A., Michael, J., Hill, F., Levy, O., and Bowman, S. R. GLUE: A multi-task benchmark and analysis platform for natural language understanding. 2019. In the Proceedings of ICLR.

Warstadt, A., Singh, A., and Bowman, S. R. Neural network acceptability judgments. *arXiv preprint 1805.12471*, 2018.

Williams, A., Nangia, N., and Bowman, S. R. A broad-coverage challenge corpus for sentence understanding through inference. In *Proceedings of NAACL-HLT*, 2018.

Yang, L. and Shami, A. On hyperparameter optimization of machine learning algorithms: Theory and practice. *Neurocomputing*, 415:295316, Nov 2020. ISSN 0925-2312. doi: 10.1016/j.neucom.2020.07.061. URL http://dx.doi.org/10.1016/j.neucom.2020.07.061.

Yeung, S., Kannan, A., Dauphin, Y., and Fei-Fei, L. Tackling over-pruning in variational autoencoders, 2017.

You, Y., Gitman, I., and Ginsburg, B. Large batch training of convolutional networks, 2017.

Yuan, K., Ying, B., and Sayed, A. On the influence of momentum acceleration on online learning. *Journal of Machine Learning Research*, 17(192):1–66, 2016.

Zagoruyko, S. and Komodakis, N. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016.

Zeiler, M. D. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

Zhang, J. and Mitliagkas, I. Yellowfin and the art of momentum tuning. *arXiv preprint arXiv:1706.03471*, 2017.