**ABSTRACT**

Momentum is a widely used technique for gradient-based optimizers in deep learning. In this paper, we propose a decaying momentum (Demon) rule. We conduct the first large-scale empirical analysis of momentum decay methods for modern neural network optimization, in addition to the most popular learning rate decay schedules. Across 28 relevant combinations of models, epochs, datasets, and optimizers, Demon achieves the highest number of Top-1 and Top-3 finishes at 39% and 85% respectively, almost doubling the second-placed learning rate cosine schedule at 17% and 60%, respectively. Demon also outperforms other widely used schedulers including, but not limited to, the learning rate step schedule, linear schedule, OneCycle schedule, and exponential schedule. Compared with the widely used learning rate step schedule, Demon is observed to be less sensitive to parameter tuning, which is critical to training neural networks in practice. Results are demonstrated across a variety of settings and architectures, including image classification, generative models, and language models. Demon is easy to implement, requires no additional tuning, and incurs almost no extra computational overhead compared to the vanilla counterparts. Code is readily available.

**KEYWORDS**

Neural networks, deep learning optimization, image classification, language models

1 INTRODUCTION

Motivation. Deep Neural Networks (DNNs) have advanced the state-of-the-art in computer vision [23, 34, 51], natural language processing [6, 19, 42] and speech recognition [54, 56], but have come with huge computation costs. A state-of-the-art language model can cost several million USD to train [9, 59]. For most practitioners, even moderate tasks can be prohibitive in time and cost when the hyperparameter tuning process is taken into account, where it is typical to retrain models many times to achieve optimal performance.

In an effort to ease the cost of training DNNs, adaptive gradient-based methods [18, 25, 32, 41, 80] were devised. Cases exist where their use leads to degraded performance [57, 75], but this can be a result of poor hyperparameter tuning [2, 12, 57, 61]. Currently, SGD with momentum (SGDM) and Adam [32] remain among the most popular methods for training DNNs. Many state-of-the-art benchmarks in Computer Vision are achieved using SGDM and a curated learning rate step schedule [23, 27, 29, 34, 51, 77, 79]. Meanwhile, variants of Adam are popular for training state-of-the-art language models [9, 16].

For optimizers to achieve good performance, their hyperparameters must be tuned properly. For example, slight changes in learning rate, learning rate decay, momentum, and weight decay (among others) can drastically alter performance. One key component to hyperparameter tuning is the selection of a good learning rate, and possibly momentum, decay schedule. Performance can vary substantially depending on this schedule, and, as we demonstrate, no one schedule is optimal. However, there is significant opportunity to improve upon existing schedules by fostering consistent state-of-the-art performance and robustness to hyperparameters across domains.

Momentum tuning. In this work, we focus on improving model performance and hyperparameter robustness with simple techniques for the momentum parameter. Momentum was designed to speed up learning in directions of low curvature, without becoming...
unstable in directions of high curvature. To minimize the objective function $L(\cdot)$, the most common momentum method, SGDM, is given by the following recursion:

$$\theta_{t+1} = \theta_t + \eta g_t,$$

for variable $\theta_t \in \mathbb{R}^p$, where $\beta$ is the momentum, $g_t$ is a stochastic gradient, where $\mathbb{E}[g_t] = \nabla L(\theta_t)$, and $\eta > 0$ is the step size.

Practitioners set $\beta = 0.9$. This is supported by recent works [11, 25, 32, 50], and by the fact that most common softwares, such as PyTorch [47], use $\beta = 0.9$ as the default momentum value. There is no indication that this choice is universally well-behaved.

There are papers that attempt to tune the momentum parameter. In the distributed setting, [44] observe that running SGD asynchronously is similar to adding a momentum-like term to SGD. They provide empirical evidence that setting $\beta = 0.9$ results in a momentum “overdose”, yielding suboptimal performance. YellowFin [81] is a learning rate and momentum adaptive method for both synchronous and asynchronous settings, motivated by a quadratic model analysis and some robustness insights. Finally, in training generative adversarial networks (GANs), optimal momentum values tend to decrease from $\beta = 0.9$ [4, 43, 48], taking even negative values [22].

This paper.} We perform the first large-scale empirical analysis of momentum decay methods and introduce the DEMON momentum decay rule, a novel method which performs favorably and increases hyperparameter robustness in comparison to other learning rate and momentum schedules. Our findings can be summarized as follows:

- We propose a new momentum decay rule, dubbed as DEMON. DEMON is motivated by decaying the total contribution of a gradient to all future updates, with limited additional computation.
- Across 28 relevant settings, DEMON achieves the highest ratio of Top-1 and Top-3 finishes: 39% and 85%, respectively. See Table 1. These ratios nearly double those of the second-placed cosine learning rate decay schedule, which achieves 17% and 60%, respectively. In addition to outperforming other popular schedule such as the learning rate step schedule, linear schedule, and OneCycle, DEMON also outperforms all other momentum schedules that were considered.
- We observe improved robustness to hyperparameter tuning for DEMON relative to the popular learning rate step schedule.

Experiments are provided on various datasets—including MNIST, FMNIST, CIFAR-10, CIFAR-100, STL-10, Tiny ImageNet, Penn Treebank (PTB), GLUE benchmark [68], and networks—including Convolutional Networks (CNN) with Residual architecture (ResNet) [23] (Wide ResNet) [79], Non-Residual architecture (VGG-16) [60], Recurrent Neural Networks (RNN) with Long Short-Term Memory architecture (LSTM) [26], Variational AutoEncoders (VAE) [33], Capsule Network [53], Noise Conditional Score Network (NCSN) [65], and BERT [16].

2 PRELIMINARIES

SGDM. Let $\theta_t \in \mathbb{R}^p$ be the parameters of the network at time step $t$, where $\eta \in \mathbb{R}$ is the learning rate, and $g_t$ is the stochastic gradient w.r.t. $\theta_t$ for empirical loss $L(\cdot)$. SGDM is parameterized by $\beta \in \mathbb{R}$, the momentum coefficient, and follows the recursion:

$$\theta_{t+1} = \theta_t + \eta g_t,$$

where $\eta_t \in \mathbb{R}^p$ accumulates momentum. Observe that for $\beta = 0$, the above recursion is equivalent to SGD. Common values for $\beta$ are closer to one, with $\beta = 0.9$ the most used value [52].

Adaptive gradient descent methods. These algorithms utilize current and past gradients to design preconditioning matrices that better approximate the local curvature of $L(\cdot)$ [18, 25]. Adam [32] uses a decaying average of past gradients, as in $E_{t+1}^g = \beta_1 \cdot E_t^g + (1 - \beta_1) \cdot g_t$, as well as a decaying average of squared gradients, as in $E_{t+1}^{g^2} = \beta_2 \cdot E_t^{g^2} + (1 - \beta_2) \cdot (g_t \circ g_t)$, leading to the recursion:

$$\theta_{t+1} = \theta_t + \eta \frac{g_t}{\sqrt{E_{t+1}^{g^2}} + \epsilon}, \ \forall t,$$

where usually $\beta_1 = 0.9$ and $\beta_2 = 0.999$.

3 DEMON: DECAYING MOMENTUM ALGORITHM

DEMON is motivated by learning rate decay models which reduce the impact of a gradient to current and future updates. By decaying momentum, we decay the total contribution of a gradient to all future updates. Our goal here is to present a concrete, effective, and easy-to-use momentum decay procedure, which we show in the.

\footnote{For clarity, we will skip the bias correction step in this description of Adam; see [32].}


**experimental section.** The key component is the momentum decay schedule:

\[
\beta_t = \beta_{\text{init}} \left(\frac{t-1}{t-\beta_{\text{init}}+\beta_{\text{init}}}\right) \cdot \left(\frac{1-\beta}{1+\beta}\right).
\]

Above, the fraction \((1 - t/T)\) refers to the proportion of iterations remaining. Fig. 1 presents a visualization of the proposed momentum decay rule and other common schedules. The interpretation of this rule comes from the following argument: Assume fixed momentum parameter \(\beta_t \equiv \beta\); e.g., \(\beta = 0.9\), as literature dictates. For our discussion, we will use the SGD recursion. We know that \(\alpha_0 = 0\), and \(\alpha_t = \beta \alpha_{t-1} - g_t\). Then, the main recursion can be unrolled into:

\[
\alpha_{t+1} = \alpha_t - \eta \nabla f_i - \eta \beta \alpha_{t-1} - \eta \beta^2 g_{t-2} + \eta \beta^3 \alpha_{t-2}
\]

where: \(\alpha_0 = 0\), and \(\alpha_t = \beta \alpha_{t-1} - g_t\). Then, the main recursion can be unrolled into:

\[
\alpha_{t+1} = \alpha_t - \eta \nabla f_i - \eta \beta \alpha_{t-1} - \eta \beta^2 g_{t-2} + \eta \beta^3 \alpha_{t-2}
\]

Introducing the above recursion, a particular gradient term \(g_t\) contributes a total of \(\eta \sum_i \beta^i\) of its “energy” to all future gradient updates. Moreover, for an asymptotically large number of iterations, we know that \(\beta\) contributes on up to \(t - 1\) terms. Then, \(\frac{\sum_{i=0}^{t-1} \beta^i}{\sum_{i=0}^{\infty} \beta^i} = \beta/(1 - \beta)\). Thus, in our quest for a simple momentum decay schedule, it is natural to consider a scheme where the cumulative momentum is decayed to 0. Let \(\beta_{\text{init}}\) be the initial \(\beta\); then at the current step \(t\) with a total of \(T\) steps, we design the decay routine such that: \(\beta/(1 - \beta) = (1 - t/T)\beta_{\text{init}}/(1 - \beta)\). This leads to (1). Although \(\beta\) changes in subsequent iterations, this is typically a very close approximation because \(\beta^i \beta^{i+1} \ldots \beta^t\) for a particular \(g^i\) diminishes much faster than \(\beta\) changes.

**Formal intuition.** Conceptually, Demon takes advantage of momentum for speed-up in the early phases of training, but decays momentum throughout training. This prevents neural network weights from growing too quickly. Constraining weights has been recently studied and shown to be key for optimal performance. In particular, it is demonstrated theoretically and empirically in [24] that plain use of momentum leads to suboptimal performance. It is important to stabilize the growth of the weights across training.

Formally, a function \(f\) is scale invariant if \(f(c \cdot x) = f(x)\) for constant \(c > 0\). Let the function \(\text{Norm}(\cdot)\) be defined as \(\text{Norm}(x) = (x - \mu_s)/\sigma_s(x)\), where \(s\) is a subset of the dimensions of \(x\), \(\mu_s\) is the mean for those dimensions, and \(\sigma_s\) the standard deviation. \(\text{Norm}(\cdot)\) is scale invariant, and includes cases such as Batch Norm [24]. Concretely, \(\text{Norm}(\theta^T x) = \text{Norm}(c \cdot\theta)^T x\). With adaptive \(\beta_t\) on iteration \(t\), we extend the norm growth lemma in [24] as follows:

**Lemma 1.** For scale invariant \(\theta\) given by momentum SGD, the following equality holds true:

\[
\|\theta_{t+1}\|^2 = \|\theta_t\|^2 + \eta^2 \|\nabla f_i\|^2 + 2\eta^2 \sum_{i=0}^{t-1} \beta_i \beta_{t-1} \ldots \beta_{t-1} \|\nabla f_i\|^2
\]

Since the latter term grows increasingly larger, Demon decays the momentum to curb its growth, and therefore reduces the norm growth of the parameters, leading to improved performance [24]. The extension is trivial and the proof is omitted.

**Connection to previous algorithms.** Demon introduces an implicit discount factor. The main recursions of the algorithm resemble those of standard machine learning algorithms. E.g., for \(\beta_t = \beta = 0.9\) we obtain SGD in Algorithm 1; in Algorithm 2, for \(\beta_t = 0.9\) with a slight adjustment of learning rate we obtain Adam, while for \(\beta_t = 0\) we obtain a non-accumulative AdaGrad algorithm. We choose to apply Demon to a slightly adjusted Adam to isolate the effect of the momentum parameter, since the momentum parameter adjusts the magnitude of the current gradient as well in vanilla Adam.

**Efficiency.** Demon requires only limited extra overhead and computation in comparison to the vanilla counterparts for the computation of \(\beta_t\). Implementation is simply 1-2 lines of code.

\[
p_t = (\text{iters} - t) / \text{iters}
\]

\[
\text{beta}_t = \text{beta}_1 \ast (p_t / (1 - \text{beta}_1 + \text{beta}_1 \ast p_t))
\]

**Convergence analysis.** We provide convergence proof for Demon SGD in the convex setting, by bounding auxiliary function sequences [20]. For an objective function \(f\) which is convex, continuously differentiable, its gradient \(\nabla f(\cdot)\) is Lipschitz continuous with constant \(L\), our goal is to show that \(f(\tilde{\theta}_T)\) converges to the optimum \(f^*\) with decreasing momentum, where \(\tilde{\theta}_T\) is the average of \(\theta_t\) for \(t = 1, ..., T\), following [20]. Our following theorem holds for a constant learning rate and \(\beta\) decaying with \(t\).

**Theorem 1.** Assume that \(f\) is convex, continuously differentiable, its gradient \(\nabla f(\cdot)\) is Lipschitz continuous with constant \(L\), with a decreasing momentum, but constant step size, as in: \(\beta_t = \beta \cdot \beta_t^{t+1/2}, \alpha \in (0, \frac{\beta}{\sqrt{T^2}})\). We consider the SGD iteration in non-stochastic settings, where: \(\theta_{t+1} = \theta_t - \alpha \nabla f(\theta_t) + \beta_t (\theta_t - \theta_{t-1})\). Then, the sequence \(\{\theta_t\}_{t=1}^T\) generated by the SGD iteration, with decreasing momentum, satisfies:

\[
f(\tilde{\theta}_T) - f^* \leq \frac{\|\theta_0 - \theta^*\|^2}{T} \frac{1}{L + \frac{\beta}{\sqrt{T^2}}}
\]

where \(\tilde{\theta}_T\) is the Cesaro average of the iterates: \(\tilde{\theta}_T = \frac{1}{T} \sum_{t=1}^T \theta_t\).

**Proof.** Let \(\beta_t = \beta \cdot \beta_t^{t+1/2}\) and \(p_t = \beta \cdot \beta_t^{t+1/2}\). We consider the SGD iteration in non-stochastic settings, where \(\theta_{t+1} = \theta_t - \alpha \nabla f(\theta_t) + \beta_t (\theta_t - \theta_{t-1})\). Using the definition of \(p_t\) above, one can easily prove that:

\[
\theta_{t+1} + p_{t+1} = \left(1 + \frac{1}{t+1}\right) \theta_{t+1} - \frac{1}{t+1} \theta_t = \theta_t + p_t - \frac{\alpha(t+2)}{t+1} \nabla f(\theta_t)
\]

Using this expression, we will analyze the term \(\|\theta_{t+1} + p_{t+1} - \theta^*\|^2:\)

\[
\|\theta_{t+1} + p_{t+1} - \theta^*\|^2 = \|\theta_t + p_t - \theta^*\|^2 - \frac{2\alpha(t+2)}{t+1} \langle \theta_t + p_t - \theta^*, \nabla f(\theta_t) \rangle + \left(\frac{\alpha(t+2)}{t+1}\right)^2 \|\nabla f(\theta_t)\|^2
\]

\[
\|\theta_t + p_t - \theta^*\|^2 - \frac{2\alpha(t+2)}{t+1} \langle \theta_t + p_t - \theta^*, \nabla f(\theta_t) \rangle - \frac{2\alpha(t+2)}{t+1} \langle \theta_t - \theta^*, \nabla f(\theta_t) \rangle + \left(\frac{\alpha(t+2)}{t+1}\right)^2 \|\nabla f(\theta_t)\|^2
\]
Since $f$ is convex, continuously differentiable, its gradient is Lipschitz continuous with constant $L$, then
\[
\frac{1}{t} \| \nabla f(\theta_t) \|^2 \leq \langle \theta_t - \theta^*, \nabla f(\theta_t) \rangle, \tag{2}
\]
\[
f(\theta_t) - f^* + \frac{1}{t} \| \nabla f(\theta_t) \|^2 \leq \langle \theta_t - \theta^*, \nabla f(\theta_t) \rangle, \tag{3}
\]
\[
f(\theta_t) - f(\theta_{t-1}) \leq \langle \theta_t - \theta_{t-1}, \nabla f(\theta_{t-1}) \rangle. \tag{4}
\]
Substituting the above inequalities leads to
\[
\| \theta_{t+1} + p_{t+1} - \theta^* \|^2 \leq \| \theta_t + p_t - \theta^* \|^2 - 2\alpha (1-\lambda)(\frac{3}{L(t+1)}) \langle f(\theta_t) - f(\theta_{t-1}) \rangle
\]
\[
- 2\alpha \lambda \frac{\| \nabla f(\theta_t) \|^2}{t+1} \leq 2\alpha \lambda \frac{\| \nabla f(\theta_t) \|^2}{t+1}.
\]
The last term is non-positive when $\alpha \in [0, \frac{t+1}{2(L^2 \lambda^2)}]$ so it can be dropped. Summing over $t = 1, ..., T$ yields
\[
2\alpha \lambda \sum_{t=1}^{T} \frac{\| \nabla f(\theta_t) \|^2}{t+1} \leq 2\alpha \lambda \sum_{t=1}^{T} \frac{\| \nabla f(\theta_t) \|^2}{t+1},
\]
implies that:
\[
2\alpha \lambda \sum_{t=1}^{T} \frac{\| \nabla f(\theta_t) \|^2}{t+1} \leq 3\alpha \langle f(\theta_t) - f^* \rangle + \| \theta_1 - \theta^* \|^2.
\]
Since $2\alpha \lambda \sum_{t=1}^{T} (f(\theta_t) - f^*) \leq 2\alpha \lambda \sum_{t=1}^{T} \frac{\| \nabla f(\theta_t) \|^2}{t+1}$, we further have:
\[
3\alpha \lambda \sum_{t=1}^{T} (f(\theta_t) - f^*) \leq \frac{3}{2} \| \theta_1 - \theta^* \|^2.
\]
Due to the convexity of $f$: $f(\theta_t) \leq \frac{1}{T} \sum_{t=1}^{T} f(\theta_t)$, observe that
\[
f(\theta_T) - f^* \leq \frac{1}{T} \sum_{t=1}^{T} (f(\theta_t) - f^*) \leq \frac{1}{3\alpha \lambda T} \left( \frac{9}{2} \alpha \| \nabla f(\theta_t) \|^2 + \frac{3}{2} \| \theta_1 - \theta^* \|^2 \right).
\]
Since $f(\theta_t) - f^* \leq \frac{1}{T} \| \theta_t - \theta^* \|^2$ by Lipschitz continuous gradients, setting $\lambda = 1$ and observing $(t+1)/(t+2) \geq 2/3$ gives the result.

For Demon Adam, we observe it lies within the definition of Generic Adam in [82], and inherits the non-convex results. Namely, to restate the theorem [82]:

**Theorem 2.** Assuming $\tau \in \{1, 2, ..., T\}$ is an index over $T$ iterations, Demon + Adam satisfies:
\[
\mathbb{E}[\| \nabla f(x_\tau) \|^2] \leq \frac{C + C'}{\alpha} \frac{\tau}{\delta \cdot \alpha T} = O \left( \frac{1}{\delta} \right),
\]
where $C, C'$ are constants, and $\alpha$ are step sizes, $\theta_t$ are the parameters related to the diagonal Hessian approximation of Adam’s preconditioner, and $\beta_t < 1$, as is the case for Demon technique. Moreover, one can remove the expectation requirement above, and have the same result hold deterministically with some probability $1 - \beta^{2/3}$:
\[
\| \nabla f(x_\tau) \|^2 \leq \frac{C + C'}{\alpha} \frac{\tau}{\delta \cdot \alpha T} = O \left( \frac{1}{\delta} \right).
\]
To achieve the above, the assumptions are lower-bounded function $f$, $L$-smoothness, and standard assumptions on gradients. Regarding the parameters, momentum $\beta_t$ has to satisfy $0 \leq \beta_t < 1$, for some $\beta$: this is explicitly the setting of Demon, where $\beta$ is the initial value of the momentum, and $\beta_t$ decreases to zero. Our work is an exact instance of decreasing momentum that leads to empirical improvements compared to previous work; to the best of our knowledge, no other paper has considered a specific decreasing schedule for momentum that is at the same time almost ‘hyperparameter-free’.

**Practical Suggestions.** We advocate for decaying momentum from $\beta_{init}$ at $t = 0$, to 0 at $t = T$ as a general rule, which is the setting we use for all Demon experiments in this paper.

**4 RELATED WORK**

Numerous techniques exist for automatic hyperparameter tuning. Adaptive methods, such as AdaGrad [18], AdaDelta [80], RMSprop [25], and Adam [32], are most widely used. Interest in closing the generalization difference between adaptive methods and SGD led to AMSGrad [50], which uses the maximum of the exponential moving average of squared gradients, QHAdam [41], a variant of QHM that recovers a variety of optimization algorithms, AdamW [38], which decouples weight decay in Adam, and Padam [10], which lowers the exponent of the second moment. YellowFin [81] is a learning rate and momentum adaptive method motivated by a quadratic model analysis and robustness insights. In the non-convex setting, STORM [13] uses a variant of momentum for variance reduction.

The convergence of momentum methods has been heavily explored both empirically and theoretically [15, 31, 71, 72, 74]. [67] explored momentum schedules, even increasing momentum during training, inspired by Nesterov’s routines for convex optimization. [63] scales the batch size to create associated changes in learning rate and momentum. [62] introduces cycles of simultaneously increasing learning rate and decreasing momentum followed by simultaneously decreasing learning rate and increasing momentum. Some work adapts the momentum to reduce oscillations during training [46] and explores integration of momentum into well-conditioned convex problems [66]. Another approach is to combine several momentum vectors with different $\beta$ values [40]. In another work, gradient staleness in variance reduction methods is addressed with gradient transport [5]. We are aware of the theoretical work of
that proves, under certain conditions, SGDM is equivalent to SGD with a rescaled learning rate, but our experiments in the deep learning setting show slightly different behavior. Understanding this discrepancy is an exciting direction of research.

Smaller values of $\beta$ have been employed for Generative Adversarial Networks (GANs), and recent developments in game dynamics [22] show a negative momentum is helpful.

5 EXPERIMENTS

Well-known experiments in the literature are selected for comparison (e.g., ResNets, LSTMs, and BERT). Training models like GPT-2/3 from scratch is not feasible, and we instead focus on providing a wide number of experiments and baselines. For each setting, we use varying numbers of epochs to demonstrate effectiveness. Experiments with different numbers of epochs are standalone experiments with independently-tuned hyperparameters. All settings are summarized in Table 2 and comprehensively detailed in Appendix A.

| Experiment short name | Model | Dataset |
|-----------------------|-------|---------|
| RN20-CIFAR10          | ResNet20 | CIFAR10 |
| RN56-TINYIMAGENET     | ResNet56 | Tiny ImageNet |
| VGG16-CIFAR100        | VGG-16 | CIFAR100 |
| WRN-STL10             | Wide ResNet 16-8 | STL10 |
| LSTM-PTB              | LSTM RNN | Penn TreeBank |
| VAE-MNIST             | VAE | MNIST |
| NCSN-CIFAR10          | CNCSN | CIFAR10 |
| CAPS-FMNIST           | Capsnet | FMNIST |
| BERTBASE-GLUE         | BERT (Pre-trained) | GLUE (9 tasks) |

We note that we have implemented setups that use the standard ImageNet dataset as input. However, we did not observe any results worth noting between learning rate step decay and DEMON. In particular, using a ResNet-50, comparable performance to state of the art is achieved using most algorithms under consideration. ImageNet results are often not indicative of generalizability, see [58], and this highlights that a breadth of experiments is important.

We consider the following schedules, where we tune both the learning rate in multiples of 3, the momentum $\epsilon \in [0,0.9,0.95,0.97]$, and weight decay, in addition to the other hyperparameters specialized to the particular schedule. The cost refers to if there are any other hyperparameters to tune that are specialized to the schedule.

- **No schedule**: Self-explanatory ($1 \times \text{cost}$).
- **Step schedule**: One of the most common kind of schedules (at this moment) for achieving state-of-the-art results in the literature [28, 29, 37, 69, 79]. We attempt decay schedules including $0.1\times$ at 50% and 75% of total epochs; $0.1\times$ at 25%, 50%, 75%; $0.1\times$ at 33% and 66%; $0.1\times$ at 10%, 25%, 50%, 75% ($4 \times \text{cost}$).
- **Cosine schedule** [39]: Follows the smooth schedule of $y_t = y_{\text{min}} + 0.5 \cdot (y_{\text{max}} - y_{\text{min}}) (1 + \cos(\pi t/T)); y$ can be the learning rate or the momentum. We consider $y_{\text{min}} = 0$ to achieve ($1 \times \text{cost}$).
- **OneCycle** [62]: This scheme roughly follows increasing learning rate linearly from $0.1\eta$ to $\eta$ and back to $0.1\eta$, while simultaneously decreasing momentum linearly from $\beta_{\text{max}}$ to $\beta_{\text{min}}$ and back to $\beta_{\text{max}}$. For the momentum values, we consider the pairs $[(0.95, 0.85), (0.9, 0.85), (0.95, 0.9)]$. For the momentum variant, we simply consider the momentum component of OneCycle ($1 \times \text{cost}$).
- **Linear schedule**: Decreases the hyperparameter from the initial value to 0 across epochs ($1 \times \text{cost}$).
- **Exponential schedule** [1]: Follows the smooth schedule of $y_t = y_0 \cdot e^{kt}$. We tune $k$ from a reasonable starting point, $-0.05\cdot(100/T)$ which is a scaled version of the curve in Figure 1 (i.e., plotted for 100 iterations in multiples of 2) ($\sim 4 \times \text{cost}$).
- **Decay on Plateau** [1]: Commonly used in practice where the learning rate is multiplied by a factor if validation loss stops improving for a certain number of epochs (patience). We tune the patience in multiples of 5, and multiply the learning rate by $0.1$ ($\sim 5 \times \text{cost}$).
- **DEMON**: This paper. The schedule follows Algorithm 1 and decays to 0 at the end of training ($1 \times \text{cost}$).

We apply these schedules to SGDM and Adam, focusing on the performance of different schedules. The overall tuning budget for DEMON SGDM/Adam is generally equal to or less than that of SGDM/Adam with its relevant possible schedules.

5.1 Decreasing the need for tuning

We demonstrate the hyperparameter robustness of DEMON SGDM and DEMON Adam relative to SGDM with learning rate step schedule and Adam. In Fig. 2, validation accuracy is displayed for a grid search over learning rate and momentum. For SGDM, results are obtained with the highest-performing learning rate schedule. The heatmaps display optimal performance of each optimizer over the full range of possible hyperparameters.

WRN-STL10-DEMONSGDM yields a significantly larger band of lighter color, indicating better performance for a wide range of hyperparameters. For every learning rate-momentum pair, we observe a lighter color for DEMON SGDM relative to SGDM. Concretely, SGDM has roughly one configuration per column with $<22\%$ generalization error, while DEMON SGDM has five.

On VGG16-CIFAR100–DEMONSGDM, a larger band of low generalization error exists compared to SGDM. There also appears to be a slight shift in optimal parameters. Concretely, DEMON SGDM has almost three times the number of configurations with generalization error $<3\%$.

On RN20-CIFAR10, DEMON Adam demonstrates its improved hyperparameter robustness relative to Adam. The generalization errors achieved with Adam fluctuate significantly, yielding optimal performance with only a few hyperparameter settings. In contrast, DEMON Adam yields a wide band of high performance across hyperparameters.

These results suggest that both DEMON Adam and DEMON SGDM are less sensitive to hyperparameter tuning than their learning rate step schedule counterparts, whilst attaining competitive error. This is critical to the use of DEMON in practice, as DEMON can yield high performance with minimal tuning. The performance of DEMON is high and stable across a wide range of hyperparameters near the default.

5.2 Results

For benchmarking purposes, we also include some other baselines where the learning rate and/or momentum are automatically
adapted. These include Quasi Hyperbolic Adam (QHAdam) [41], Quasi Hyperbolic Momentum (QHM) [41], AMSGrad [30], AdamW [38], YellowFin [81], Aggregated Momentum (AggMo) [40]. Quasi Hyperbolic methods are capable of recovering Accelerated SGD [30], Nesterov Accelerated Gradient [45], Synthesized Nesterov Variants [35], and others, thus covering more algorithms than those present. However, these methods are included primarily for reference, and the major focus of this work is on the schedules describe at the beginning of Section 5. We emphasize that DEMON can be combined with any momentum method. We present results with SGDM and Adam due to their wide usage.

Mainline results We summarize the results of all relevant settings in Table 1. Out of the 28 relevant settings in this paper, DEMON achieves the highest percentage of Top-1 and Top-3 finishes, with 39% and 85% respectively. Other momentum decay schedules are not competitive, likely due to being overly aggressive. Learning rate step schedule, linear schedule, and cosine schedule perform comparably; however, the different working schedules do not always yield comparable performance across settings. For example, linear learning rate decay performs exceptionally well in the ResNet settings, but closer to average on other settings. Such results indicate that the decay schedule, for learning rate or momentum, is an additional hyperparameter that must be tuned. Even though Decay On Plateau and Exponential Decay are implemented in most popular frameworks as empirical alternatives, they perform poorly when the total number of epochs are predefined. DEMON is the most consistent across a wide range of settings, with by far the highest Top-3 performance.

Residual Networks (RN28–CIFAR10). We train a ResNet20 [23] model on the CIFAR-10 dataset. We emphasize that ResNet20 is commonly conflated with the more expressive ResNet18, which achieves different performance. This is an important setting to evaluate because ResNets remain one of the most popular computer vision architectures in both academia and industry, achieving reasonable performance with less risk of overfitting to the test set [58]. The other ResNet settings include WRN-STL10, a Wide Residual 16-8 model [79] on STL-10 that has significantly fewer, higher resolution images in comparison to CIFAR, and RN56-TINYIMAGENET, a ResNet56 model on the Tiny ImageNet dataset. Due to limited resources, the runs of RN56–TINYIMAGENET are conducted to supplement the other settings. See Table 3 for results.

Non-Residual Convolutional Network (VGG16–CIFAR10). We train an adjusted VGG-16 model [60] on the CIFAR-100 dataset. Previous observations from the ResNet settings continue to hold, where other momentum decay schedules do not perform well, likely due to the overly aggressive momentum decay. The momentum component of OneCycle, when used in isolation, appears to destabilize training, leading in some cases to performance worse than the vanilla counterpart. Whilst traditionally the learning rate step schedule is the most popular and often used to achieve state-of-the-art results in computer vision, this schedule has no clear advantage over the learning rate cosine schedule, the learning rate linear schedule, or DEMON. DEMON has the strongest performance, even outperforming methods that automatically tune the momentum parameter.
Table 3: RN20–CIFAR10, WRN–STL10 and RN56–TINYIMAGENET generalization error. The number of epochs was predefined before the execution of the algorithms. Red indicates Top-1 performance, bold is Top-3, ignoring non SGDM and Adam optimizers.

| ResNet 20 | Wide ResNet 16-8 | ResNet 56 |
|-----------|------------------|-----------|
| **SGDM**  | 75 epochs | 150 epochs | 300 epochs | 50 epochs | 100 epochs | 300 epochs | 50 epochs | 100 epochs | 300 epochs |
| + LR Step Schedule | 8.82 ± 2.2 | 8.43 ± 0.7 | 7.32 ± 0.1 | 22.42 ± 0.56 | 17.20 ± 0.35 | 14.51 ± 0.26 | 45.98 ± 0.23 | 41.66 ± 0.10 |
| + LR Cosine Schedule | 8.80 ± 0.8 | 8.10 ± 0.13 | 7.78 ± 0.14 | 20.03 ± 0.26 | 17.02 ± 0.24 | 14.66 ± 0.25 | - | - |
| + OneCycle | 10.83 ± 2.5 | 9.23 ± 0.19 | 8.42 ± 0.12 | 21.67 ± 0.27 | 19.69 ± 0.21 | 19.00 ± 0.42 | - | - |
| + LR Linear Schedule | 9.03 ± 2.4 | 8.13 ± 0.12 | 7.62 ± 0.12 | 19.54 ± 0.20 | 17.39 ± 0.24 | 14.58 ± 0.18 | - | - |
| + LR Decay on Plateau | 9.05 ± 0.7 | 8.26 ± 0.07 | 7.97 ± 0.14 | 21.05 ± 0.27 | 18.73 ± 0.39 | 15.16 ± 0.36 | - | - |
| + LR Exp decay | 9.55 ± 0.9 | 9.20 ± 0.23 | 7.82 ± 0.05 | 22.65 ± 0.49 | 20.60 ± 0.21 | 15.85 ± 0.28 | - | - |
| + OneCycle Momentum | 15.61 ± 39 | 14.02 ± 13 | 13.35 ± 5.8 | 29.34 ± 7.8 | 23.20 ± 39 | 25.42 ± 47 | - | - |
| + Cosine Momentum | 13.57 ± 20 | 11.23 ± 22 | 10.87 ± 0.03 | 24.12 ± 0.34 | 21.66 ± 0.37 | 16.29 ± 0.26 | - | - |
| + Linear Momentum | 12.31 ± 14 | 10.26 ± 16 | 10.63 ± 31 | 25.13 ± 0.28 | 22.74 ± 0.75 | 17.92 ± 0.41 | - | - |
| +Exp Momentum decay | 14.96 ± 19 | 12.98 ± 15 | 12.35 ± 11 | 24.01 ± 0.20 | 19.35 ± 29 | 17.56 ± 21 | - | - |
| **AggMo + LR Step** | 8.71 ± 2.4 | 7.93 ± 1.5 | 7.62 ± 0.03 | 21.37 ± 0.32 | 17.15 ± 0.35 | 14.49 ± 0.26 | - | - |
| **QHM + LR Step** | 8.72 ± 1.4 | 7.95 ± 1.7 | 7.67 ± 1.0 | 21.75 ± 0.31 | 18.21 ± 0.48 | 14.44 ± 0.23 | - | - |
| **DEMON SGDM** | 8.56 ± 1.0 | 8.21 ± 1.8 | 7.59 ± 1.2 | 20.23 ± 0.31 | 16.19 ± 0.23 | 14.44 ± 0.53 | 44.87 ± 0.15 | 40.85 ± 0.01 |

| Adam | 13.63 ± 2.2 | 11.90 ± 0.6 | 11.94 ± 0.06 | 23.35 ± 0.29 | 19.63 ± 0.26 | 18.65 ± 0.07 | 57.56 ± 1.50 | 50.89 ± 0.59 |
| + LR Step Schedule | 10.47 ± 1.0 | 8.75 ± 0.17 | 8.55 ± 0.05 | 23.85 ± 0.07 | 19.63 ± 0.33 | 18.29 ± 0.10 | - | - |
| + LR Cosine Schedule | 9.56 ± 1.2 | 9.15 ± 0.12 | 8.93 ± 0.07 | 22.85 ± 0.47 | 21.47 ± 0.31 | 19.08 ± 0.36 | - | - |
| + OneCycle | 10.33 ± 2.0 | 9.87 ± 0.12 | 9.03 ± 0.18 | 20.02 ± 0.19 | 19.21 ± 0.28 | 19.03 ± 0.43 | - | - |
| + LR Linear Schedule | 9.25 ± 1.2 | 9.20 ± 0.22 | 8.89 ± 0.05 | 21.70 ± 0.11 | 21.53 ± 0.44 | 17.85 ± 0.15 | - | - |
| + LR Decay on Plateau | 9.71 ± 3.9 | 8.92 ± 0.18 | 8.80 ± 0.11 | 22.77 ± 0.33 | 19.91 ± 0.45 | 19.61 ± 0.56 | - | - |
| + LR Exp decay | 10.48 ± 1.5 | 9.24 ± 0.16 | 8.53 ± 0.07 | 23.30 ± 0.39 | 20.70 ± 0.50 | 19.63 ± 0.24 | - | - |
| + OneCycle Momentum | 20.05 ± 9.1 | 15.60 ± 0.69 | 14.85 ± 0.50 | 24.61 ± 0.54 | 23.39 ± 0.39 | 25.34 ± 0.38 | - | - |
| + Cosine Momentum | 11.08 ± 11.10 | 0.36 ± 0.20 | 10.64 ± 0.30 | 25.76 ± 0.22 | 23.58 ± 0.02 | 20.10 ± 0.15 | - | - |
| + Linear Momentum | 11.91 ± 18.11 | 14.38 ± 0.13 | 11.09 ± 0.12 | 24.36 ± 0.31 | 21.93 ± 0.23 | 21.81 ± 0.36 | - | - |
| + Exp Momentum decay | 15.18 ± 0.10 | 12.08 ± 0.16 | 10.63 ± 0.12 | 28.90 ± 0.21 | 25.28 ± 0.31 | 22.90 ± 0.41 | - | - |
| **A MSRGrad** | 13.43 ± 1.4 | 11.83 ± 0.12 | 10.48 ± 0.12 | 21.73 ± 0.25 | 19.35 ± 0.20 | 18.21 ± 0.18 | - | - |
| **AdamW** | 12.46 ± 0.52 | 11.38 ± 0.21 | 10.50 ± 0.17 | 20.39 ± 0.62 | 18.55 ± 0.23 | 17.00 ± 0.41 | - | - |
| **QHadam** | 15.55 ± 0.25 | 13.78 ± 0.08 | 13.36 ± 0.11 | 21.25 ± 0.22 | 19.81 ± 0.18 | 18.52 ± 0.25 | - | - |
| **YellowFin** | 13.66 ± 0.34 | 12.13 ± 0.41 | 11.39 ± 0.16 | 22.55 ± 0.14 | 20.68 ± 0.04 | 18.56 ± 0.33 | - | - |
| **DEMON Adam** | 9.68 ± 0.7 | 8.90 ± 0.18 | 8.50 ± 0.12 | 20.95 ± 0.23 | 19.50 ± 0.32 | 18.62 ± 0.41 | 48.92 ± 0.03 | 45.72 ± 0.31 |

Figure 4: Randomly selected CIFAR10 images generated with NCSN. Left: Real CIFAR10 images. Middle: Adam. Right: DEMON Adam.

advantage over the cosine schedule, linear schedule, or DEMON, the latter of which performs the best. See Table 4.

**LSTM (PTB–LSTM).** We apply an LSTM [26] architecture to the language modeling task, which has notoriously sharp gradient distributions (e.g., rare words). We use the official TensorFlow v1 implementation for PTB - LSTM. OneCycle Momentum is adjusted to decay to 0 and back since this setting typically requires low momentum to train well. Such characteristically low momentum makes this task a difficult test case for momentum decay methods, as momentum seems to have less impact on performance relative to other settings. However, more momentum values are also swept to achieve the reported perplexity. As a whole, smooth decay methods do not perform comparably to step decay methods on this task. DEMON is surprisingly competitive and is the only momentum
Table 4: VGG16-CIFAR100 generalization error, LSTM-PTB generalization perplexity, VAE-MNIST generalization loss, and CAPS-FMNIST generalization error. The number of epochs was predefined before the execution of the algorithms. Red indicates Top-1 performance, bold is Top-3, ignoring non SGDM and Adam optimizers.

| Algorithm       | VGG-16  | LSTM  | VAE    | CAPSNET |
|-----------------|---------|-------|--------|---------|
| SGDM            |         |       |        |         |
| 150 epochs      | 300 epochs | 25 epochs | 39 epochs | 50 epochs | 100 epochs | 50 epochs | 100 epochs |
| + LR Step Schedule | 30.09 ± 32 | 27.83 ± 30 | 81.67 ± .21 | 82.02 ± .13 | 140.28 ± 51 | 137.70 ± .93 | - | - |
| + LR Cosine Schedule | 28.63 ± 11 | 27.84 ± 12 | 81.64 ± .37 | 83.23 ± .06 | 139.15 ± 26 | 136.69 ± .27 | - | - |
| + OneCycle      | 30.10 ± 34 | 29.09 ± 12 | 90.03 ± .39 | 91.19 ± .01 | 139.79 ± 66 | 137.20 ± .06 | - | - |
| + LR Linear Schedule | 29.10 ± 34 | 28.26 ± 08 | 96.27 ± .09 | 98.79 ± .02 | 148.00 ± .48 | 141.72 ± .48 | - | - |
| + LR Decay on Plateau | 30.65 ± 31 | 29.74 ± .43 | 81.53 ± .24 | 81.82 ± .07 | 140.51 ± .73 | 139.54 ± .34 | - | - |
| + LR Exp decay  | 29.51 ± 22 | 28.57 ± .18 | 84.30 ± .08 | 83.49 ± .03 | 154.31 ± .43 | 145.83 ± .48 | - | - |
| + OneCycle Momentum | 35.86 ± 25 | 35.34 ± .30 | 87.14 ± .27 | 91.93 ± 1.03 | 144.90 ± .61 | 142.63 ± .25 | - | - |
| + Cosine Momentum | 32.73 ± 07 | 30.99 ± .11 | 88.33 ± .92 | 90.02 ± .10 | 145.15 ± .85 | 140.14 ± .81 | - | - |
| + Linear Momentum | 31.61 ± 29 | 31.23 ± .26 | 90.02 ± .51 | 93.06 ± .29 | 145.33 ± .13 | 139.83 ± .14 | - | - |
| + Exp Momentum decay | 34.50 ± 23 | 32.83 ± .15 | 86.39 ± .20 | 89.45 ± .63 | 150.54 ± 1.07 | 156.95 ± .47 | - | - |
| AggMo + LR Step | 30.75 ± 55 | 28.64 ± .45 | 83.15 ± .12 | 83.43 ± .17 | 139.49 ± .99 | 136.56 ± .28 | - | - |
| QHM + LR Step   | 29.93 ± 13 | 29.01 ± 54 | 88.75 ± .23 | 88.42 ± .10 | 142.47 ± 50 | 137.97 ± .54 | - | - |
| AggMo + QHM + LR Step | 30.67 ± 11 | 27.69 ± 11 | 82.66 ± .05 | 84.84 ± .22 | 138.29 ± 0.8 | 136.55 ± .64 | - | - |

Table 5: NCSN-CIFAR10 Inception score. The number of epochs was predefined before the execution of the algorithms.

| Algorithm   | NCSN (512 epochs) |
|-------------|-------------------|
| Adam        | 8.15 ± 0.20       |
| DEMON Adam  | 8.07 ± 0.08       |

Variational AutoEncoder (VAE-MNIST). Generative modeling is a branch of unsupervised learning that focuses on learning the underlying data distribution. VAEs [33] are generative models that pair a generator network with a recognition model that performs approximate inference and can be trained with backprop. We train VAEs on MNIST. General trends follow the ResNet settings. Interestingly, the learning rate linear schedule perform poorly with SGDM, but are improved in the Adam setting. See Table 4.

Capsule Network (CAPS-FMNIST). Capsule Networks [53] represent Neural Networks as a set of capsules that each encode a specific entity or meaning. Capsules exploit the observation that viewpoint changes significantly alter pixels but are linear with respect to the pose matrix. The activation of capsules differs from standard neural network activation functions because it depends on comparing incoming pose predictions. We train Capsule Networks on the FMNIST dataset with only Adam and its variants, which are typically used in this setting[53]. Highlighting the unpredictability of the performance among learning rate schedules, the exponential learning rate decay schedule is unremarkable in other settings, but is clearly the best learning rate schedule in this setting. See Table 4.

Noise Conditional Score Network (NCSN-CIFAR10). NCSN [65] is a recent generative model that estimates gradients of the data distribution with score matching and produces samples via Langevin dynamics. We train a NCSN on CIFAR10, for which NCSN achieves a strong inception score. Although vanilla Adam achieves a slightly superior inception score (see Table 5) the results in Figure 4 are unnaturally green compared to those produced by DEMON Adam.
Table 6: Results of BERT_{BASE/GLUE}. Adam + LR Linear Schedule follows the huggingface [76] implementation, and achieves the results in well-known studies [16, 55].

| Score | CoLA | MNLI | MRPC | QNLI | QQP | RTE | SST-2 | STS-B | WNLI |
|-------|------|------|------|------|-----|-----|-------|-------|------|
| Adam + LR Linear Schedule | 79.1 | 57.4 | 84.3 | 89.0 | 91.4 | 89.2 | 69.4 | 92.7 | 89.0 |
| Demon Adam | 79.7 | 58.4 | 84.2 | 90.0 | 90.9 | 89.0 | 69.4 | 92.5 | 88.8 |

Table 7: VGG16-CIFAR100-DEMONSGDM, RN20-CIFAR10-DEMONSGDM generalization error, PTB-LSTM-DEMONSGDM (perplexity) and VAE-MNIST-DEMONSGDM generalization loss. The number of epochs was predefined before the execution.

| | RN-20 | VGG-16 | LSTM | VAE |
| | 150 epochs | 300 epochs | 50 epochs | 100 epochs |
| SGD ELR | 9.14 ± 24 | 8.58 ± 08 | 8.16 ± 15 | 7.59 ± 12 |
| Demon SGM | 8.71 ± 24 | 7.95 ± 15 | 7.59 ± 12 | 7.32 ± 21 |

BERT (BERT_{BASE/GLUE}). BERT [16] is one of the most influential language models in the last few years. The key characteristic of BERT is the ability for a pre-trained model to be fine-tuned to achieve strong performance across a large variety of language tasks. The GLUE benchmark [68] is a collection of nine different language tasks [3, 7, 8, 14, 17, 21, 36, 49, 64, 70, 73], and is a common benchmark in NLP. We achieve the performance in well-known studies [16, 55], using the huggingface [76] default implementation of Adam with learning rate linear decay, tuning only the learning rate. We also only tune the learning rate for Demon Adam. Results are given in Table 6. Running the same seeds, Demon Adam yields a slight improvement over the baseline.

6 DEMON AND EFFECTIVE LEARNING RATE

We present results of Demon against the effective learning rate adjusted SGD (SGD ELR). The effective learning rate is proposed to approximate SGD with ELR, where the learning rate is adjusted with a factor of 1/(1 − m) and m is the momentum coefficient. However, the results in Table 7 demonstrate that Demon cannot be accurately approximated with an effective learning rate adjusted SGD. For settings (PTB-LSTM-DEMONSGDM and VAE-MNIST-DEMONSGDM), SGD ELR causes learning to diverge. In Table 7, there exists a 0.5-3% generalization error gap for VGG16-CIFAR100 and for RN20-CIFAR10.

7 CONCLUSION

We show the effectiveness of Demon across a large number of datasets and architectures. We demonstrate that Demon can be effectively applied to both SGD and Adam. Compared to other learning rate schedules and momentum schedules, Demon achieves the largest number of Top-1 and Top-3 finishes. This includes improvements over the popular learning rate step schedule, cosine decay schedule, OneCycle, and many others. Demon is computationally cheap, understandable, and easy to implement. We hope it is useful in practice and as a subject of future research.

REFERENCES

[1] Martin Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Jianmin Kal, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2016. TensorFlow: A system for large-scale machine learning. In 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16). 265–283. https://www.usenix.org/system/files/conference/osdi16/osdi16-abadi.pdf

[2] Naman Agarwal, Rohan Anil, Eldad Hazan, Tomer Koren, and Cyril Zhang. 2020. Revisiting the Generalization of Adaptive Gradient Methods. (2020).

[3] Eneko Agirre, Liu M. Arnaud, and Richard Wicentowski (Eds.). 2007. Proceedings of the Fourth International Workshop on Semantic Evaluations (SemEval-2007). Association for Computational Linguistics, Prague, Czech Republic.

[4] Martin Arjovsky, Soumith Chintala, and Léon Bottou. 2017. Wasserstein GAN. arXiv:1701.07875 [stat.ML].

[5] Sebastien Arnold, Pierre-Antoine Manzagol, Reza Behanerazeh, Ioannis Mitliagkas, and Nicolas Le Roux. 2019. Reducing the variance in online optimization by transporting past gradients. In Advances in Neural Information Processing Systems. 5392–5403.

[6] Dimitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. arXiv preprint arXiv:1409.0473 (2014).

[7] Roy Bar Haim, Ido Dagan, Bill Dolan, Lisa Ferro, Danilo Giampiccolo, Bernardo Magnini, and Idan Spector. 2006. The second PASCAL recognising textual entailment challenge. (2006).

[8] Luisa Bentivogli, Ido Dagan, Hoa Trang Danu, Danilo Giampiccolo, and Bernardo Magnini. 2009. The Fifth PASCAL Recognizing Textual Entailment Challenge. (2009).

[9] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Arizel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language Models are Few-Shot Learners. arXiv:2005.14165 [cs.CL].

[10] Jinghui Chen and Quanquan Gu. 2018. Closing the generalization gap of adaptive gradient methods in training deep neural networks. arXiv preprint arXiv:1806.06763 (2018).

[11] Jianmin Chen, Xinghao Pan, Rajat Monga, Samy Bengio, and Rafal Jozefowicz. 2016. Revisiting distributed synchronous SGD. arXiv preprint arXiv:1604.00981 (2016).

[12] Dami Choi, Christopher J. Shallue, Zachary Nado, Jaehoon Lee, Chris J. Maddison, and G. Dahl. 2019. On Empirical Comparisons of Optimizers for Deep Learning. arXiv abs/1910.05446 (2019).

[13] Ashok Cutkosky and Francesco Orabona. 2019. Momentum-Based Variance Reduction in Non-Convex SGD. arXiv preprint arXiv:1905.10018 (2019).

[14] Ido Dagan, Oren Glickman, and Bernardo Magnini. 2006. The PASCAL recognising textual entailment challenge. In Machine learning challenges. evaluating predictive uncertainty, visual object classification, and recognising tectual entailment. Springer, 177–190.

[15] Aaron Defazio and Robert M. Gower. 2020. Factorial Powers for Stochastic Optimization. arXiv:2006.01244 [cs.LG].

[16] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. arXiv:1810.04805 [cs.CL].

[17] William B Dolan and Chris Brockett. 2005. Automatically constructing a corpus of sentential paraphrases. In Proceedings of the International Workshop on Paraphrasing.
[73] Adina Williams, Nikita Nangia, and Samuel R. Bowman. 2018. A Broad-Coverage Challenge Corpus for Sentence Understanding through Inference. In Proceedings of NAACL-HLT.

[74] Ashia C. Wilson, Benjamin Recht, and Michael I Jordan. 2016. A Lyapunov analysis of momentum methods in optimization. arXiv preprint arXiv:1611.02635 (2016).

[75] Ashia C Wilson, Rebecca Roelofs, Mitchell Stern, Nati Srebro, and Benjamin Recht. 2017. The marginal value of adaptive gradient methods in machine learning. In Advances in Neural Information Processing Systems. 4148–4158.

[76] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Pujol, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. 2020. HuggingFace’s Transformers: State-of-the-art Natural Language Processing. arXiv:1910.03771 [cs.CL]

[77] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. 2017. Aggregated residual transformations for deep neural networks. In Proceedings of the IEEE conference on computer vision and pattern recognition. 1492–1500.

[78] Kun Yuan, Bicheng Ying, and Ali Sayed. 2016. On the influence of momentum acceleration on online learning. Journal of Machine Learning Research 17, 192 (2016), 1–66.

[79] Sergey Zagoruyko and Nikos Komodakis. 2016. Wide residual networks. arXiv preprint arXiv:1605.07146 (2016).

[80] Matthew D Zeiler. 2012. ADADelta: an adaptive learning rate method. arXiv preprint arXiv:1212.5701 (2012).

[81] Jian Zhang and Ioannis Mitliagkas. 2017. Yellowfin and the art of momentum tuning. arXiv preprint arXiv:1706.03471 (2017).

[82] Fangyu Zou, Li Shen, Zequn Jie, Weizhong Zhang, and Wei Liu. 2018. A sufficient condition for convergences of Adam and RMSProp. arxiv preprint arXiv:1811.09358 (2018).
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 [23] 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 [23] 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 [60] 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 [79] 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 [26] 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 [53] 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 [33] 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 [65] 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 [68] consists of 9 different language tasks [3, 7, 8, 14, 17, 21, 36, 49, 64, 70, 73], grouped together to form a benchmark. BERT [16] 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.