DEMON: MOMENTUM DECAY FOR IMPROVED NEURAL NETWORK TRAINING

Anonymous authors
Paper under double-blind review

ABSTRACT

Momentum is a popular technique in deep learning for gradient-based optimizers. We propose a decaying momentum (DEMON) rule, motivated by decaying the total contribution of a gradient to all future updates. Applying DEMON to Adam leads to significantly improved training, notably competitive to momentum SGD with learning rate decay, even in settings in which adaptive methods are typically non-competitive. Similarly, applying DEMON to momentum SGD improves over momentum SGD with learning rate decay in most cases. Notably, DEMON momentum SGD is observed to be less sensitive to parameter tuning than momentum SGD with learning rate decay schedule, 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 and tune, and incurs limited extra computational overhead, compared to the vanilla counterparts. Code is readily available.

1 INTRODUCTION

Motivation. Deep Neural Networks (DNNs) have advanced the state-of-the-art in computer vision (Krizhevsky et al., 2012; He et al., 2016; Ren et al., 2015), natural language processing (Mikolov et al., 2013; Bahdanau et al., 2014; Gehring et al., 2017) and speech recognition (Sak et al., 2014; Sercu et al., 2016), but have come with huge computation costs. A state-of-the-art language model can cost several million USD to train (Brown et al., 2020; Sharir et al., 2020). For most practitioners and researchers, 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 (Duchi et al., 2011; Zeiler, 2012; Hinton et al., 2012; Kingma & Ba, 2014; Ma & Yarats, 2018) were devised. However, there are cases where their use leads to a performance gap (Wilson et al., 2017; Shah et al., 2018). Although this performance gap was shown to be a result of poor hyperparameter tuning (Sivaprasad et al., 2019; Choi et al., 2019; Agarwal et al., 2020; Shah et al., 2018), stochastic gradient descent (SGD) and SGD with momentum (SGDM) remain among the most popular methods for training DNNs. In fact, SGDM remains the optimizer of choice and state-of-the-art performance on many benchmarks—such as the image classification dataset ImageNet—is produced with SGDM (Krizhevsky et al., 2012; He et al., 2016; Xie et al., 2017; Zagoruyko & Komodakis, 2016; Huang et al., 2017; Ren et al., 2015; Howard et al., 2017).

For optimizers, including SGDM, to achieve good performance, their hyperparameters must be tuned properly. Slight changes in learning rate, learning rate decay, momentum, and weight decay (amongst others) can drastically alter performance. Hyperparameter tuning is extremely time consuming, and researchers often resort to a costly grid search. *Thus, the holy grail is achieving the performance and efficiency of SGDM with decreased dependence on hyperparameter tuning.*

Momentum tuning. The focus of this work is on how we can boost performance and practically reduce dependency on hyperparameter tuning with a simple technique 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 $\mathcal{L}(\cdot)$, the most common momentum method, SGDM, is given by the following recursion for variable vector $\theta_t \in \mathbb{R}^p$:

$$\theta_{t+1} = \theta_t + \eta v_t, \quad v_t = \beta v_{t-1} - g_t.$$

where β controls the rate of momentum decay, g_t represents a stochastic gradient, usually $\mathbb{E}[g_t] = \nabla \mathcal{L}(\theta_t)$, and $\eta > 0$ is the step size.

Practitioners usually set $\beta = 0.9$. This setting is supported by recent works that prescribe it (Chen et al., 2016; Kingma & Ba, 2014; Hinton et al., 2012; Reddi et al., 2019), and by the fact that most common softwares, such as PyTorch (Paszke et al., 2017), 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, (Mitliagkas et al., 2016) 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. Additionally, YellowFin (Zhang & Mitliagkas, 2017) 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 $\beta = 0.9$ (Mirza & Osindero, 2014; Radford et al., 2015; Arjovsky et al., 2017), taking even negative values (Gidel et al., 2018).

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 significantly surpasses the performance of both Adam and SGDM in their current form (and other state-of-the-art methods), while empirically increasing robustness to hyperparameters. Our findings can be summarized as follows:

- We conduct the first large-scale empirical analysis of momentum decay methods for modern neural network optimization. We propose a new momentum decay rule DEMON, which performs well empirically and is motivated by decaying the total contribution of a gradient to all future updates, with limited overhead and additional computation.
- Adding the momentum decay rule to vanilla Adam, we observe large performance gains. The network continues to learn for far longer after Adam begins to plateau.
- We observe improved performance for SGDM with momentum decay over learning rate decay; an interesting result given the unparalleled effectiveness of learning rate decay.

Experiments are provided on various datasets—including MNIST, FMNIST, CIFAR-10, CIFAR-100, STL-10, Tiny ImageNet, Penn Treebank (PTB); and networks—including Convolutional Networks (CNN) with Residual architecture (ResNet) (He et al., 2016) (Wide ResNet) (Zagoruyko & Komodakis, 2016), Non-Residual architecture (VGG-16) (Simonyan & Zisserman, 2014), Recurrent Neural Networks (RNN) with Long Short-Term Memory architecture (LSTM) (Hochreiter & Schmidhuber, 1997), Variational AutoEncoders (VAE) (Kingma & Welling, 2015), Capsule Network (Sabour et al., 2017), Noise Conditional Score Network (NCSN) (Song & Ermon, 2019).

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. θ_t for empirical loss $\mathcal{L}(\cdot)$, such that $\mathbb{E}[g_t] = \nabla \mathcal{L}(\theta_t)$. SGDM is parameterized by $\beta \in \mathbb{R}$, the momentum coefficient, and follows the recursion:

$$\theta_{t+1} = \theta_t + \eta v_t, \quad v_t = \beta v_{t-1} - g_t,$$

where $v_t \in \mathbb{R}^p$ accumulates momentum. Observe that for $\beta = 0$, the above recursion is equivalent to SGD. Common values for β are closer to one, with $\beta = 0.9$ the most used value (Ruder, 2016).

Adaptive gradient descent methods. These algorithms utilize current and past gradient information to design preconditioning matrices that better approximate the local curvature of $\mathcal{L}(\cdot)$ (Duchi et al., 2011; Hinton et al., 2012). On top of SGDM, Adam (Kingma & Ba, 2014) uses an exponentially decaying average of past gradients, as in $\mathcal{E}^g_{t+1} = \beta_1 \cdot \mathcal{E}^g_t + (1 - \beta_1) \cdot g_t$, as well as a decaying average of squared gradients, as in $\mathcal{E}^{g \circ g}_{t+1} = \beta_2 \cdot \mathcal{E}^{g \circ g}_t + (1 - \beta_2) \cdot (g_t \circ g_t)$, leading to the recursion:[1]

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{\mathcal{E}^{g \circ g}_{t+1,i} + \varepsilon}} \cdot \mathcal{E}^g_{t+1,i}, \quad \forall t,$$

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

[1]For clarity, we will skip the bias correction step in this description of Adam; see Kingma & Ba (2014).

3 DEMON: DECAYING MOMENTUM ALGORITHM

Algorithm 1 DEMON in SGDM

Parameters: T, η, initial mom. β_{init}.
$v_0 = \theta_0 = 0$ or random.
for $t = 0, \ldots, T$ **do**
$\beta_t = \beta_{\text{init}} \cdot \frac{\left(1-\frac{t}{T}\right)}{(1-\beta_{\text{init}})+\beta_{\text{init}}\left(1-\frac{t}{T}\right)}$
$\theta_{t+1} = \theta_t - \eta g_t + \beta_t v_t$
$v_{t+1} = \beta_t v_t - \eta g_t$
end for

Algorithm 2 DEMON in Adam

Parameters: T, η, initial mom. β_{init}, β_2, $\varepsilon = 10^{-8}$. $v_0 = \theta_0 = 0$ or random.
for $t = 0, \ldots, T$ **do**
$\beta_t = \beta_{\text{init}} \cdot \frac{\left(1-\frac{t}{T}\right)}{(1-\beta_{\text{init}})+\beta_{\text{init}}\left(1-\frac{t}{T}\right)}$
$\mathcal{E}^{g \circ g}_{t+1} = \beta_2 \cdot \mathcal{E}^{g \circ g}_{t} + (1-\beta_2) \cdot (g_t \circ g_t)$
$m_{t,i} = g_{t,1} + \beta_t m_{t-1,i}$
$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{\mathcal{E}^{g \circ g}_{t+1,i} + \varepsilon}} \cdot m_{t,i}$
end for

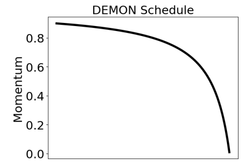

Figure 1: This plot illustrates the *non-linear* DEMON schedule, decaying from $\beta_{init} = 0.9$ to 0.

Motivation and interpretation. DEMON is motivated by learning rate decay models which reduce the impact of a gradient to current and future updates. By decaying the momentum parameter, 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 experimental section.* The key component is the momentum decay schedule:

$$\beta_t = \beta_{\text{init}} \cdot \frac{\left(1-\frac{t}{T}\right)}{(1-\beta_{\text{init}})+\beta_{\text{init}}\left(1-\frac{t}{T}\right)} \,. \tag{1}$$

Above, the fraction $(1 - t/T)$ refers to the proportion of iterations remaining. Fig. 1 presents a visualization of the proposed momentum decay rule. Only the DEMON momentum schedule is presented in the main text, but comparisons to other possible momentum schedules are presented in Appendix G. We use the DEMON momentum schedule due to its superior empirical performance in comparison to other options for momentum decay. 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 SGDM recursion. We know that $v_0 = 0$, and $v_t = \beta v_{t-1} - g_t$. Then, the main recursion can be unrolled into:

$$\theta_{t+1} = \theta_t - \eta g_t - \eta\beta g_{t-1} - \eta\beta^2 g_{t-2} + \eta\beta^3 v_{t-2} = \cdots = \theta_t - \eta g_t - \eta \cdot \sum_{i=1}^{t} \left(\beta^i \cdot g_{t-i}\right)$$

Interpreting 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 β contributes on up to $t-1$ terms. Then, $\sum_{i=1}^{\infty} \beta^i = \beta \sum_{i=0}^{\infty} \beta^i = \beta/(1-\beta)$. Thus, in our quest for a decaying schedule and for a simple momentum decay, it is natural to consider a scheme where the cumulative momentum is decayed to 0. Let β_{init} be the initial β; then at current step t with total T steps, we design the decay routine such that: $\beta/(1-\beta) = (1-t/T)\beta_{\text{init}}/(1-\beta_{\text{init}})$. This leads to equation 1. Although β changes in subsequent iterations, this is typically a very close approximation since $\beta^i\beta^{i+1}\ldots\beta^t$ for a particular g^i diminishes much faster than β changes. We emphasize that directly decaying β linearly achieves significantly worse performance (see Appedix

Connection to previous algorithms. DEMON introduces an implicit discount factor. The main recursions of the algorithm are the same with standard algorithms in machine learning. E.g., for $\beta_t = \beta = 0.9$ we obtain SGD with momentum, and for $\beta = 0$ we obtain plain SGD in Algorithm 1; in Algorithm 2, for $\beta_1 = 0.9$ with a slightly adjustment of learning rate we obtain Adam, while for $\beta_1 = 0$ we obtain a non-accumulative AdaGrad algorithm. We choose to apply DEMON to a slightly adjusted Adam—instead of vanilla 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 β_t. Implementation is simply 1-2 lines of code.

```
p_t = (iters - t) / iters
beta_t = beta1 * (p_t / (1 - beta1 + beta1 * p_t))
```

Convergence analysis. *We provide convergence proof for* DEMON *SGDM in the convex setting, by bounding auxiliary function sequences as in (Ghadimi et al., 2014).* Convergence results for DEMON Adam in the non-convex setting can be easily extended from existing work Zou et al. (2018). See

Appendix C for details. Convergence analysis on a toy convex problem on accelerated SGD as in (Kidambi et al., 2018) is given in Appendix D.

Practical suggestions. For settings in which β_{init} is typically large, such as image classification, we advocate for decaying momentum from β_{init} at $t = 0$, to 0 at $t = T$ as a general rule. We also observe and report improved performance by delaying momentum decay till later epochs.

4 RELATED WORK

Numerous techniques exist for automatic hyperparameter tuning. Adaptive methods, such as AdaGrad (Duchi et al., 2011), AdaDelta (Zeiler, 2012), RMSprop (Hinton et al., 2012), and Adam (Kingma & Ba, 2014), are most widely used. Interest in closing the generalization difference between adaptive methods and SGDM led to AMSGrad (Reddi et al., 2019), which uses the maximum of the exponential moving average of squared gradients, QHAdam (Ma & Yarats, 2018), a variant of QHM that recovers a variety of optimization algorithms, AdamW (Loshchilov & Hutter, 2017), which decouples weight decay in Adam, and Padam (Chen & Gu, 2018), which lowers the exponent of the second moment. YellowFin (Zhang & Mitliagkas, 2017) is a learning rate and momentum adaptive method motivated by a quadratic model analysis and robustness insights. In the non-convex setting, STORM (Cutkosky & Orabona, 2019) uses a variant of momentum for variance reduction.

The convergence of momentum methods has been heavily explored both empirically and theoretically (Wibisono & Wilson, 2015; Wibisono et al., 2016; Wilson et al., 2016; Kidambi et al., 2018; Defazio & Gower, 2020). Sutskever et al. (2013) explored momentum schedules, even increasing momentum during training, inspired by Nesterov's routines for convex optimization. Smith et al. (2017) scales the batch size to create associated changes in learning rate and momentum. Smith (2018) 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 (Odonoghue & Candes, 2015) and explores integration of momentum into well-conditioned convex problems (Srinivasan et al., 2018). Another approach is to combine several momentum vectors with different β values (Lucas et al., 2018). In another work, gradient staleness in variance reduction methods is addressed with gradient transport Arnold et al. (2019). We are aware of the theoretical work of Yuan et al. (2016) 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 β have been employed for Generative Adversarial Networks (GANs), and recent developments in game dynamics (Gidel et al., 2018) show a negative momentum is helpful.

5 EXPERIMENTS

We separate experiments into those with adaptive learning rate and those with adaptive momentum, following Ma & Yarats (2018). Well-known experiments in the literature are selected for comparison (e.g., ResNets and LSTMs). *We avoided training pre-trained models as it is less informative (for optimization purposes) than training models from scratch. 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 regardless of training budget. Experiments with different numbers of epochs are standalone experiments with independently-tuned hyperparameters. All settings are briefly summarized in Table 1 and comprehensively detailed, including exact hyperparameters, in Appendix A. Plots are in Appendix F.

Table 2 presents hyperparameter tuning details for SGDM, Adam, and DEMON variants (see Table 6 for exact details). The exploration setting (CIFAR10/100/STL10, and following general guidelines in literature) informed our settings for other experiments. **The overall tuning budget for SGDM in exploration was 2-3x that of DEMON SGDM, and is the same for mainline experiments**.

5.1 DECREASING THE NEED FOR HYPERPARAMETER TUNING

In this section, we demonstrate the robustness of DEMON SGDM and DEMON Adam relative to SGDM with learning rate decay 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 $< 31\%$.

On RN20-CIFAR10-DEMONAdam, 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.

Table 1: Summary of experimental settings.

Experiment short name	Model	Dataset	Optimizer
RN20-CIFAR10-DEMONSGDM	ResNet20	CIFAR10	DEMON SGDM
RN20-CIFAR10-DEMONAdam	ResNet20	CIFAR10	DEMON Adam
RN56-TINYIMAGENET-DEMONSGDM	ResNet56	Tiny ImageNet	DEMON SGDM
RN56-TINYIMAGENET-DEMONAdam	ResNet56	Tiny ImageNet	DEMON Adam
VGG16-CIFAR100-DEMONSGDM	VGG-16	CIFAR100	DEMON SGDM
VGG16-CIFAR100-DEMONAdam	VGG-16	CIFAR100	DEMON Adam
WRN-STL10-DEMONSGDM	Wide ResNet 16-8	STL10	DEMON SGDM
WRN-STL10-DEMONAdam	Wide ResNet 16-8	STL10	DEMON Adam
LSTM-PTB-DEMONSGDM	LSTM RNN	Penn TreeBank	DEMON SGDM
LSTM-PTB-DEMONAdam	LSTM RNN	Penn TreeBank	DEMON Adam
VAE-MNIST-DEMONSGDM	VAE	MNIST	DEMON SGDM
VAE-MNIST-DEMONAdam	VAE	MNIST	DEMON Adam
NCSN-CIFAR10-DEMONAdam	NCSN	CIFAR10	DEMON Adam
CAPS-FMNIST-DEMONAdam	Capsnet	FMNIST	DEMON Adam

Table 2: Hyperparameter tuning details for main optimizers. DEMON schedule tuning refers to beginning at 0%, 50%, or 75% of epochs.

	Mainline settings		Exploration settings	
Optimizer	Tuned	Fixed	Tuned	Fixed
SGDM	η	β, η Schedule	η, β, η Schedule	
Adam	η	β, β_2	η, β	β_2
DEMONSGDM	η	β_{init}, Demon schedule	η, β_{init}, Demon schedule	
DEMONAdam	η	β_{init}, β_2, Demon schedule	η, β_{init}	β_2, Demon schedule

These results indicate that both DEMON Adam and SGDM are less sensitive to hyperparameter tuning than their vanilla counterparts, whilst attaining same or better 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 COMPARISON OF ADAPTIVE METHODS

DEMON Adam (Algorithm 2) is applied to a variety of models and tasks. We select Adam (Kingma & Ba, 2014) as the baseline algorithm and include comparisons with state-of-the-art methods Quasi-Hyperbolic Adam (QHAdam) (Ma & Yarats, 2018), AMSGrad (Reddi et al., 2019), AdamW (Loshchilov & Hutter, 2017), YellowFin (Zhang & Mitliagkas, 2017) (descriptions in Appendix A.2.1). We emphasize that Quasi-Hyperbolic methods are capable of recovering Accelerated SGD (Jain et al., 2017), Nesterov Accelerated Gradient (Nesterov, 1983), Synthesized Nesterov Variants (Lessard et al., 2016), and others, thus covering more algorithms than those present. We tune all learning rates in rough multiples of 3 and keep all other parameters close to those recommended in the original literature. For DEMON Adam, we set $\beta_{\text{init}} = 0.9, \beta_2 = 0.999$ and decay from β_{init} to 0 starting from 75% of epochs.

To simplify the results presentation, further comparison with Padam (Chen & Gu, 2018) and OneCycle (Smith, 2018) are in Appendix H. DEMON outperforms both approaches in various settings.

Residual Network (`RN20-CIFAR10-DEMONAdam`). We train a ResNet20 (He et al., 2016) model on the CIFAR-10 dataset. With DEMON Adam, we achieve the generalization error reported in the literature (He et al., 2016), attained using SGDM and a curated learning rate decay schedule, while all other adaptive methods are not competitive (see Table 3). DEMON Adam continues learning after other methods have plateaued, outperforming other adaptive methods by a 2%-5% generalization error across all experiments (see Appendix F Figure 5).

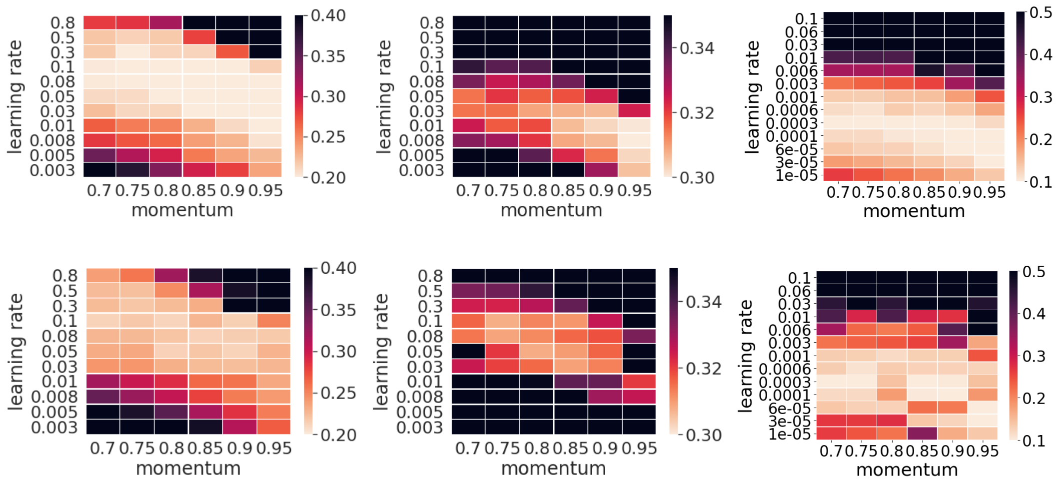

Figure 2: In order from left to right: Error rate for `WRN-STL10-DEMONSGDM` (top) and `WRN-STL10-SGDM` (bottom) for 50 epochs, error rate for `VGG16-CIFAR100-DEMONSGDM` (top) and `VGG16-CIFAR100-SGDM` (bottom) for 100 epochs, and error rate for `RN20-CIFAR10-DEMONAdam` and `RN20-CIFAR10-Adam` for 100 epochs. Light-colored patches indicate better performance.

Table 3: `RN20-CIFAR10-DEMONSGDM/Adam` and `VGG16-CIFAR100-DEMONSGDM/Adam` generalization error. The number of epochs was predefined before the execution of the algorithms.

	ResNet 20				**VGG-16**		
	30 epochs	75 epochs	150 epochs	300 epochs	75 epochs	150 epochs	300 epochs
SGDM	11.80 ± .11	8.82 ± .25	8.43 ± .07	**7.32** ± .14	35.04 ± .24	30.09 ± .32	27.83 ± .30
AggMo	11.14 ± .34	8.71 ± .24	7.93 ± .15	7.62 ± .03	34.40 ± .60	30.75 ± .55	28.64 ± .45
QHM	10.66 ± .17	8.72 ± .14	7.95 ± .17	7.67 ± .10	33.27 ± .56	29.93 ± .13	29.01 ± .54
Adam	16.58 ± .18	13.63 ± .22	11.90 ± .06	11.94 ± .06	37.98 ± .20	33.62 ± .11	31.09 ± .09
AMSGrad	16.98 ± .36	13.43 ± .14	11.83 ± .12	10.48 ± .12	40.67 ± .65	34.46 ± .21	31.62 ± .12
AdamW	15.39 ± .46	12.46 ± .52	11.38 ± .21	10.50 ± .17	36.96 ± 1.21	33.48 ± .68	32.22 ± .13
QHAdam	16.41 ± .38	15.55 ± .25	13.78 ± .08	13.36 ± .11	36.53 ± .20	32.96 ± .11	30.97 ± .10
YellowFin	17.25 ± .15	13.66 ± .34	12.13 ± .41	11.39 ± .16	86.24 ± 3.54	68.87 ± 5.82	50.18 ± 4.02
DEMON SGDM	**10.78** ± .08	**8.49** ± .19	**7.63** ± .17	**7.32** ± .19	**31.62** ± .29	**28.19** ± .11	27.68 ± .24
DEMON Adam	11.75 ± .15	9.69 ± .10	8.83 ± .08	8.44 ± .05	32.40 ± .19	28.84 ± .18	**27.11** ± .19

(`RN56-TINYIMAGENET-DEMONAdam`). We train a ResNet56 (He et al., 2016) model on the Tiny ImageNet dataset. Using DEMON Adam, we achieve superior generalization performance over Adam, which was found to be quite sensitive to hyperparameter settings. DEMON Adam outperformed Adam on Tiny ImageNet by a 5%-8% margin in generalization error (see Table 5).

Non-Residual Network (`VGG16-CIFAR100- DEMONAdam`). We train an adjusted VGG-16 model (Simonyan & Zisserman, 2014) on the CIFAR-100 dataset. Again, DEMON Adam continues to improve after other methods begin to plateau, resulting in a 1-3% decrease in generalization error compared to reported results with the same model and task (Sankaranarayanan et al., 2018), attained with SGDM and a curated learning rate decay schedule. DEMON Adam achieves a 3%-6% generalization error improvement over all other methods (see Appendix F Figure 5 and Table 3).

Wide Residual Network (`WRN-STL10-DEMONAdam`). We train a Wide Residual 16-8 model (Zagoruyko & Komodakis, 2016) on the STL-10 dataset, which has significantly fewer, higher resolution images in comparison to CIFAR. In this setting, DEMON Adam significantly outperforms other methods in the later stages of training. DEMON Adam achieves a 0.5%-2% generalization error margin over other methods with varying epochs (see Appendix F Figure 5 and Table 4).

Capsule Network (`CAPS-FMNIST-DEMONAdam`). Capsule Networks (Sabour et al., 2017) 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 and show that DEMON Adam outperforms other methods (see Table 4).

LSTM (`PTB-LSTM-DEMONAdam`). We apply an LSTM (Hochreiter & Schmidhuber, 1997) to the language modeling task, which can have sharp gradient distributions (e.g., rare words). While all other adaptive methods overfit to this task, DEMON Adam achieves a *margin of 6-14 in generalization perplexity* (see Appendix F Figure 5 and Table 4).

Variational AutoEncoder (`VAE-MNIST-DEMONAdam`). Generative modeling is a branch of unsupervised learning that focuses on learning the underlying data distribution. VAEs (Kingma & Welling, 2015) 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. DEMON Adam outperforms all other methods (see Table 5, Appendix F Figure 5).

Table 4: `WRN-STL10-DEMONSGDM/Adam` generalization error, `PTB-LSTM-DEMONSGDM/Adam` generalization perplexity, and `CAPS-FMNIST-DEMONSGDM/Adam` generalization error. The number of epochs was predefined before the execution of the algorithms.

	Wide Residual 16-8			**LSTM**		**Capsule Network**	
	50 epochs	100 epochs	200 epochs	25 epochs	39 epochs	50 epochs	100 epochs
SGDM	22.42 ± .56	17.20 ± .35	14.51 ± .26	89.59 ± .07	**87.57** ± .11	-	-
AggMo	21.37 ± .32	17.15 ± .35	14.49 ± .26	89.09 ± .16	89.07 ± .15	-	-
QHM	21.75 ± .31	18.21 ± .48	14.44 ± .23	94.47 ± .19	94.44 ± .13	-	-
Adam	23.35 ± .20	19.63 ± .26	18.65 ± .07	115.54 ± .64	115.02 ± .52	9.27 ± .08	9.25 ± .11
AMSGrad	21.73 ± .25	19.35 ± .20	18.21 ± .18	108.07 ± .19	107.87 ± .25	9.39 ± .18	9.28 ± .19
AdamW	20.39 ± .62	18.55 ± .23	17.00 ± .41	116.27 ± 2.57	116.21 ± 2.14	9.78 ± 0.62	9.92 ± .74
QHAdam	21.25 ± .22	19.81 ± .18	18.52 ± .25	112.52 ± .23	112.45 ± .39	9.30 ± .23	9.24 ± .15
YellowFin	22.55 ± .14	20.68 ± .04	18.56 ± .33	123.52 ± .52	115.55 ± .23	10.96 ± .65	10.55 ± .84
DEMON SGDM	**19.40** ± .08	**16.07** ± .28	**13.59** ± .67	**88.33** ± .16	88.32 ± .12	-	-
DEMON Adam	19.42 ± .10	17.82 ± .34	16.87 ± .36	101.57 ± .32	101.44 ± .47	**8.82** ± .12	**8.76** ± .13

Table 5: `RN56-TINYIMAGENET-DEMONSGDM/Adam` generalization error, `VAE-MNIST-DEMONSGDM/Adam` generalization loss, and `NCSN-CIFAR10-DEMONAdam` inception score. The number of epochs was predefined before the execution of the algorithms.

	Resnet 56		**VAE**			**NCSN**
	20 epochs	40 epochs	50 epochs	100 epochs	200 epochs	512 epochs
SGDM	45.98 ± .21	41.66 ± .10	140.28 ± .51	137.70 ± .93	136.34 ± .31	-
AggMo	-	-	139.49 ± .99	136.56 ± .28	134.93 ± .40	-
QHM	-	-	142.47 ± .50	137.97 ± .54	135.97 ± .29	-
Adam	57.56 ± 1.50	50.89 ± .59	136.28 ± .18	134.64 ± .14	134.66 ± .17	**8.15** ± .20
AMSGrad	-	-	137.89 ± .12	135.69 ± .03	134.75 ± .18	-
QHAdam	-	-	136.69 ± .17	134.84 ± .08	134.12 ± .12	-
YellowFin	-	-	414.74 ± 5.00	351.80 ± 6.68	286.69 ± 6.68	-
DEMON SGDM	**44.87** ± .15	**40.85** ± .01	138.29 ± .08	136.55 ± .64	134.68 ± .30	-
DEMON Adam	48.92 ± .03	45.72 ± .31	**134.46** ± .17	**134.12** ± .08	**133.87** ± .21	8.07 ± .08

Noise Conditional Score Network (`NCSN-CIFAR10- DEMONAdam`). NCSN (Song & Ermon, 2019) 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 Adam achieves a superior inception score (see Table 5), the results in Figure 3 are unnaturally green compared to those produced by DEMON Adam.

5.3 NON-ADAPTIVE MOMENTUM METHODS

We apply DEMON SGDM (Algorithm 1) to a variety of models and tasks. SGDM with learning rate decay is the baseline because it often achieves state-of-the-art results for such tasks. Experiments with recent adaptive momentum methods Aggregated Momentum (AggMo) (Lucas et al., 2018) and Quasi-Hyperbolic Momentum (QHM) (Ma & Yarats, 2018) are included (descriptions in Appendix A.2.2). We exclude accelerated SGD (Jain et al., 2017) due to tuning difficulties. We tune all learning rates in rough multiples of 3 and keep other parameters close to values recommended in the literature. For SGDM, AggMo, and QHM, we decay the learning rate at 50% and 75% of total epochs by a factor of 0.1, except for the `PTB-LSTM-DEMONSGDM` setting. For DEMON SGDM, we apply no learning rate decay, start decay at 75% of epochs, $\beta_{\text{init}} = 0.95$ for all experiments and decay to 0. Unlike the other methods, for a particular model-dataset DEMON SGDM did not require any retuning for different number of total epochs.

Figure 3: Randomly selected CIFAR10 images generated with NCSN. Left: Real CIFAR10 images. Middle: Adam. Right: DEMON Adam.

We emphasize that DEMON *can be combined with any momentum method. We present* DEMON *SGDM since SGDM is the most widely used optimizer.*

Residual Network (`RN20-CIFAR10-DEMONSGDM`). We train a ResNet20 model on the CIFAR-10 dataset. With DEMON SGDM, we achieve better generalization error than SGDM. Additionally, DEMON SGDM outperforms other adaptive momentum methods with learning rate decay schedules (see Table 3), demonstrating the strong performance of DEMON relative to learning rate decay.

(`RN56-TINYIMAGENET-DEMONSGDM`). We train a ResNet56 model on the Tiny ImageNet dataset. DEMON SGDM outperforms SGDM with learning rate decay (see Table 5). The performance achieved with SGDM was found to be sensitive to hyperparameter settings, while DEMON SGDM performed comparably for a wide range of hyperparameters.

Non-Residual Network (`VGG16-CIFAR100-DEMONSGDM`). We train an adjusted VGG-16 model on the CIFAR-100 dataset. DEMON SGDM learns slowly in initial epochs but continues to learn after the performance of other methods plateaus, yielding superior final performance. DEMON SGDM improves on all other methods in generalization error margin (see Table 3).

Wide Residual Network (`WRN-STL10-DEMONSGDM`). We train a Wide Residual 16-8 model on the STL-10 dataset. DEMON SGDM outperforms all other methods by a 1%-2% generalization error margin with a small and large number of epochs (see Table 4).

LSTM (`PTB-LSTM-DEMONSGDM`). We train an LSTM architecture for the PTB language modeling task. DEMON SGDM slightly outperforms other adaptive momentum methods in generalization perplexity and is competitive with SGDM (see Table 4).

Variational AutoEncoder (`VAE-MNIST-DEMONSGDM`). We train a generative VAE model on the MNIST dataset. DEMON SGDM outperforms all other methods in terms of generalization loss for fewer epochs and is competitive when more epochs are used (see Table 5).

6 ADDITIONAL EXPERIMENTAL RESULTS

Additional experimental results are given in the Appendix and we provide summaries in this section. In Appendix H, we demonstrate superior performance of DEMON SGDM over Padam Chen & Gu (2018) and OneCycle Smith (2018) on `RN20-CIFAR10-DEMONSGDM`, `VGG16-CIFAR100-DEMONSGDM` and `VAE-MNIST-DEMONSGDM` settings. In Appendix E, we demonstrate the superior performance DEMON SGDM over an effective learning rate adjusted SGD. In Appendix I, we run all experiments from Section 5.3 without learning rate decay. On RN20-CIFAR10-DEMONSGDM, DEMON SGDM outperforms all other adaptive momentum methods by a 3%-8% validation error margin with a small and large number of epochs. On VGG16-CIFAR100-DEMONSGDM, DEMON SGDM achieves a 1%-8% improvement in generalization error over all other methods.

7 CONCLUSION

We show the effectiveness of DEMON across a number of datasets and architectures. The adaptive optimizer Adam combined with DEMON is empirically far superior to the popular Adam, in addition to other state-of-the-art algorithms, suggesting a drop-in replacement. It is also demonstrated that DEMON SGDM improves upon SGDM with learning rate decay schedule. Furthermore, DEMON is empirically robust to hyperparameter tuning. DEMON is computationally cheap, understandable, and easy to implement. We hope it is useful in practice and a subject of future research.

REFERENCES

Naman Agarwal, Rohan Anil, Elad Hazan, Tomer Koren, and Cyril Zhang. Revisiting the generalization of adaptive gradient methods. 2020.

Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein GAN. *arXiv preprint arXiv:1701.07875*, 2017.

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

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel 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. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

Jianmin Chen, Xinghao Pan, Rajat Monga, Samy Bengio, and Rafal Jozefowicz. Revisiting distributed synchronous SGD. *arXiv preprint arXiv:1604.00981*, 2016.

Jinghui Chen and Quanquan Gu. Closing the generalization gap of adaptive gradient methods in training deep neural networks. *arXiv preprint arXiv:1806.06763*, 2018.

Dami Choi, Christopher J. Shallue, Zachary Nado, Jaehoon Lee, Chris J. Maddison, and G. Dahl. On empirical comparisons of optimizers for deep learning. *ArXiv*, abs/1910.05446, 2019.

Ashok Cutkosky and Francesco Orabona. Momentum-based variance reduction in non-convex sgd. *arXiv preprint arXiv:1905.10018*, 2019.

Aaron Defazio and Robert M. Gower. Factorial powers for stochastic optimization, 2020.

Timothy Dozat. Incorporating nesterov momentum into adam. *ICLR Workshop, (1):20132016*, 2016.

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

Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N Dauphin. Convolutional sequence to sequence learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 1243–1252. JMLR. org, 2017.

Euhanna Ghadimi, Hamid Reza Feyzmahdavian, and Mikael Johansson. Global convergence of the heavyball method for convex optimization. *arXiv preprint arXiv:1412.7457*, 2014.

Gauthier Gidel, Reyhane Askari Hemmat, Mohammad Pezeshki, Remi Lepriol, Gabriel Huang, Simon Lacoste-Julien, and Ioannis Mitliagkas. Negative momentum for improved game dynamics. *arXiv preprint arXiv:1807.04740*, 2018.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

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

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

Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

Jie Hu, Li Shen, Samuel Albanie, Gang Sun, and Enhua Wu. Squeeze-and-excitation networks. *arxiv preprint arXiv:1709.01507*, 2017.

Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4700–4708, 2017.

Prateek Jain, Sham M. Kakade, Rahul Kidambi, Praneeth Netrapalli, and Aaron Sidford. Accelerating stochastic gradient descent for least squares regression. *arXiv preprint arXiv:1704.08227*, 2017.

Rahul Kidambi, Praneeth Netrapalli, Prateek Jain, and Sham Kakade. On the insufficiency of existing momentum schemes for stochastic optimization. In *2018 Information Theory and Applications Workshop (ITA)*, pp. 1–9. IEEE, 2018.

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

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

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pp. 1097–1105, 2012.

Laurent Lessard, Benjamin Recht, and Andrew Packard. Analysis and design of optimization algorithms via integral quadratic constraints. *SIAM Journal on Optimization*, 26(1):57–95, 2016.

Tsung-Yi Lin, Piotr Dollar, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. *CVPR*, 2017.

Ilya Loshchilov and Frank Hutter. Fixing weight decay regularization in adam. *arXiv preprint arXiv:1711.05101*, 2017.

James Lucas, Shengyang Sun, Richard Zemel, and Roger Grosse. Aggregated momentum: Stability through passive damping. *arXiv preprint arXiv:1804.00325*, 2018.

Jerry Ma and Denis Yarats. Quasi-hyperbolic momentum and Adam for deep learning. *arXiv preprint arXiv:1810.06801*, 2018.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pp. 3111–3119, 2013.

Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*, 2014.

Ioannis Mitliagkas, Ce Zhang, Stefan Hadjis, and Christopher Ré. Asynchrony begets momentum, with an application to deep learning. In *2016 54th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pp. 997–1004. IEEE, 2016.

Yurii Nesterov. A method for solving the convex programming problem with convergence rate of (1/k2). *Soviet Mathematics Doklady*, 27(2):372–376, 1983.

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

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

Sashank J Reddi, Satyen Kale, and Sanjiv Kumar. On the convergence of Adam and beyond. *arXiv preprint arXiv:1904.09237*, 2019.

Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pp. 91–99, 2015.

Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

Sara Sabour, Nicholas Fross, and Geoffrey Hinton. Dynamic routing between capsules. In *Advances in neural information processing systems*, 2017.

Haşim Sak, Andrew Senior, and Françoise Beaufays. Long short-term memory recurrent neural network architectures for large scale acoustic modeling. In *Fifteenth annual conference of the international speech communication association*, 2014.

Swami Sankaranarayanan, Arpit Jain, Rama Chellappa, and Ser Nam Lim. Regularizing deep networks using efficient layerwise adversarial training. *arXiv preprint arXiv:1705.07819*, 2018.

Tom Sercu, Christian Puhrsch, Brian Kingsbury, and Yann LeCun. Very deep multilingual convolutional neural networks for LVCSR. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 4955–4959. IEEE, 2016.

Vatsal Shah, Anastasios Kyrillidis, and Sujay Sanghavi. Minimum norm solutions do not always generalize well for over-parameterized problems. *arXiv preprint arXiv:1811.07055*, 2018.

Or Sharir, Barak Peleg, and Yoav Shoham. The cost of training nlp models: A concise overview. *arXiv preprint arXiv:2004.08900*, 2020.

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

Prabhu Sivaprasad, Florian Mai, Thijs Vogels, Martin Jaggi, and Franois Fleuret. On the tunability of optimizers in deep learning. 10 2019.

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

Samuel Smith, Pieter-Jan Kindermans, Chris Ying, and Quoc Le. Don't decay the learning rate, increase the batch size. *arXiv preprint arXiv:1711.00489*, 2017.

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

Vishwak Srinivasan, Adepu Ravi Sankar, and Vineeth N Balasubramanian. Adine: an adaptive momentum method for stochastic gradient descent. In *Proceedings of the ACM India Joint International Conference on Data Science and Management of Data*, pp. 249–256. ACM, 2018.

Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *International conference on machine learning*, pp. 1139–1147, 2013.

Fei Wang, Mengqing Jiang, Chen Qian, Shuo Yang, Cheng Li, Honggang Zhang, Xiaogang Wang, and Xiaoou Tang. Residual attention network for image classification. *CVPR*, 2017.

Andre Wibisono and Ashia C Wilson. On accelerated methods in optimization. *arXiv preprint arXiv:1509.03616*, 2015.

Andre Wibisono, Ashia C Wilson, and Michael I Jordan. A variational perspective on accelerated methods in optimization. *proceedings of the National Academy of Sciences*, 113(47):E7351–E7358, 2016.

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

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

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

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

Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016.

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

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

Fangyu Zou, Li Shen, Zequn Jie, Weizhong Zhang, and Wei Liu. A sufficient condition for convergences of adam and rmsprop. *arxiv preprint arXiv:1811.09358*, 2018.