Decaying momentum helps neural network training

John Chen JOHNCHEN@RICE.EDU
Anastasios Kyrillidis ANASTASIOS@RICE.EDU
Rice University, Houston, TX 77005

Abstract

Momentum is a simple and 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 rivals momentum SGD with learning rate decay, and in some cases leads to improved performance. DEMON is trivial to implement and incurs limited extra computational overhead, compared to the vanilla counterparts.

1. Introduction

Deep Neural Networks (DNNs) have drastically advanced the state-of-the-art performance in many computer science applications, including computer vision [10, 19, 30], natural language processing [1, 6, 24] and speech recognition [31, 33]. Yet, in the face of such significant developments, the age-old stochastic gradient descent (SGD), and the accelerated variant SGD with momentum (SGDM), algorithm remains one of the most, if not the most, popular method for training DNNs [9, 38, 42].

Adaptive methods [5, 11, 17, 23, 46] sought to simplify the training process, while providing similar performance. However, while they are often used by practitioners, there are cases where their use leads to a performance gap [34, 42]. At the same time, much of the state-of-the-art performance on highly contested benchmarks—such as the image classification dataset ImageNet—have been produced with SGDM [10, 13, 14, 19, 30, 43, 45].

Nevertheless, a key factor in any algorithmic success still lies in hyperparameter tuning. For example, in the literature above, they obtain such performance with a well-tuned SGD with momentum and a learning rate decay schedule, or with a proper hyperparameter tuning in adaptive methods. Slight changes in learning rate, learning rate decay, momentum, and weight decay (amongst others) can drastically alter performance. Hyperparameter tuning is arguably one of the most time consuming parts of training DNNs, and researchers often resort to a costly grid search. *Thus, finding new and simple hyper-parameter tuning routines that boost the performance of state of the art algorithms is of ultimate importance and one of the most pressing problems in machine learning.*

The focus of this work is on the momentum parameter and how we can boost the performance of training methods with a simple technique. Momentum helps speed up learning in directions of low curvature, without becoming unstable in directions of high curvature. Minimizing the objective function $\mathcal{L}(\cdot)$, the simplest and most common momentum method, SGDM, is given by the following recursion for variable vector $\theta_t \in \mathbb{R}^p$:

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

The coefficient β—traditionally, selected constant in $[0, 1]$—controls how quickly the momentum decays, g_t represents a stochastic gradient, usually $\mathbb{E}[g_t] = \nabla \mathcal{L}(\theta_t)$, and $\eta > 0$ is the step size.

But how do we select β? The most prominent choice among practitioners is $\beta = 0.9$. This is supported by recent works that prescribe it [2, 11, 17, 29], and by the fact that most common softwares, such as PyTorch [28], declare $\beta = 0.9$ as the default value in their optimizer implementations. However, there is no indication that this choice is universally well-behaved.

In this work, we introduce a novel momentum decay rule which significantly surpasses the performance of both Adam and SGDM (as they are used currently), in addition to other state-of-the-art adaptive learning rate and adaptive momentum methods, across a variety of datasets and networks. In particular, our findings can be summarized as follows:

$i)$ We propose a new momentum decay rule, motivated by decaying the total contribution of a gradient to all future updates, with limited overhead and additional computation.

$ii)$ Using the momentum decay rule with Adam, we observe large performance gains—relative to vanilla Adam—where the network continues to learn for far longer after Adam begins to plateau, and suggest that the momentum decay rule should be used as default for this method.

$iii)$ We observe comparative performance for SGDM between momentum decay and learning rate decay; an interesting result given the unparalleled effectiveness of learning rate decay schedule.

Experiments are provided on various datasets, including MNIST, CIFAR-10, CIFAR-100, STL-10, Penn Treebank (PTB), and networks, including Convolutional Neural Networks (CNN) with Residual architecture (ResNet) [10], Wide Residual architecture (Wide ResNet) [45], Non-Residual architecture (VGG-16) [35], Recurrent Neural Networks (RNN) with Long Short-Term Memory architecture (LSTM) [12], Variational AutoEncoders (VAE) [18], and the recent Noise Conditional Score Network (NCSN) [36].

2. Demon: Decaying momentum algorithm

Algorithm 1 Demon in SGDM

```
Parameters: # of iterations $T$, step size $\eta$, momentum initial value $\beta_{\text{init}}$.
$v_t = \theta_t = 0$, otherwise randomly initialized.
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: # of iterations T, step size $\eta$, momentum initial value $\beta_{\text{init}}$, $\beta_2$, $\varepsilon = 10^{-8}$.
$v_t = \theta_t = \mathcal{E}_0^{g \circ g} = 0$, otherwise randomly initialized.
for $t = 0, \ldots, T$ do
    $\beta_t = \beta_{\text{init}} \cdot \frac{\left(1 - \frac{t}{T}\right)}{(1-\beta_{\text{init}}) + \beta_{\text{init}}\left(1 - \frac{t}{T}\right)}$
    $\mathcal{E}_{t+1}^{g \circ g} = \beta_2 \cdot \mathcal{E}_t^{g \circ g} + (1 - \beta_2) \cdot (g_t \circ g_t)$
    $m_{t,i} = g_{t,1} + \beta_t m_{t-1,i}$
    $\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{\mathcal{E}_{t+1,i}^{g \circ g} + \varepsilon}} \cdot m_{t,i}$
end for
```

Motivation and interpretation. Demon is motivated by learning rate rules: by decaying the momentum parameter, we decay the total contribution of a gradient to all future updates. Similar reasoning applies for learning rate decay routines: however, *our goal here is to present a concrete and easy-to-use momentum decay procedure, which can be used with or without learning rate routines, as we show in the experimental section.* The key component is the momentum decay schedule:

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

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 accelerated SGD recursion. We know that $v_0 = 0$, and $v_t = \beta v_{t-1} - \eta g_{t-1}$. 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 linear 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.

Connection to other 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 2; 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.

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. In many cases, performance can be further improved by decaying to a small negative value, such as 0.3.

3. Related work

There are numerous techniques for automatic hyperparameter tuning. The most widely used are learning rate adaptive methods, starting with AdaGrad [5], AdaDelta [46], RMSprop [11], and Adam [17]. Adam [17], the most popular, introduced a momentum term, which is combined with the current gradient before multiplying with an adaptive learning rate. Interest in closing the generalization difference between adaptive methods and SGDM led to AdamW [21], by fixing the weight decay of Adam, and Padam [3], by lowering the exponent of the second moment.

Asynchronous methods are commonly used in deep learning, and [25] show that running SGD asynchronously is similar to adding a momentum-like term to SGD without assumptions of convexity of the objective function. YellowFin [47] is a learning rate and momentum adaptive method for both the synchronous and asynchronous setting motivated by a quadratic model analysis and robustness insights.

There is substantial research, both empirical and theoretical, into the convergence of momentum methods [16, 39–41]. In addition, [38] explored momentum schedules, with even increasing momentum schedules during training, inspired by Nesterov's routines for convex optimization. There is some work into reducing oscillations during training, by adapting the momentum [27]. There is also work into adapting momentum in well-conditioned convex problems as opposed to setting to zero [37]. Another approach in this area is to keep several momentum vectors according to different β and combining them [22]. We are aware of the theoretical work of [44] which prove under certain conditions that momentum SGD is equivalent to SGD with a rescaled learning rate, however our experiments in the deep learning setting show slightly different behavior and understanding why is an exciting direction of research.

Smaller values of β have gradually been employed for Generative Adversarial Networks (GAN), and recent developments in game dynamics [8] show a negative momentum is helpful for GANs.

4. Experiments

We separate experiments into those with adaptive learning rate and those with adaptive momentum. We report improved performance by delaying the application of DEMON where applicable, and report performance across different number of total epochs to demonstrate effectiveness regardless of the training budget. Note that the predefined number of epochs run affects the proposed decaying momentum routine, by definition of β_t. We run 5 seeds.

4.1. Adaptive methods

At first, we apply DEMON Adam (Algorithm 2) to a variety of models and tasks. We select vanilla Adam as the baseline algorithm and include more recent state-of-the-art adaptive learning rate methods Quasi-Hyperbolic Adam (QHAdam) [23] and AMSGrad [29] in our comparison. See Appendix B.2.1 for details. We tune all learning rates in roughly multiples of 3 and try to keep all other parameters close to those recommended in the original literature. For DEMON Adam, we leave $\beta_{\text{init}} = 0.9, \beta_2 = 0.999$ and decay from β_{init} to 0 in all experiments.

We report results in tabulated form in Tables 1, 2 and Figure 1, with images produced by generative networks in the Appendix B.3. DEMON Adam appears to continue to learn after other methods have plateaued, leading to significantly better performance. With DEMON Adam applied to a ResNet18 network on CIFAR10, we achieve the generalization error reported in the literature [10] for this model, attained using SGDM and a curated learning rate decay schedule. With DEMON Adam applied to a VGG16 network on CIFAR100, we observe a 1-3% decrease in generalization error than typically reported results with the same model and task [32], which are attained using SGDM and a curated learning rate decay schedule.

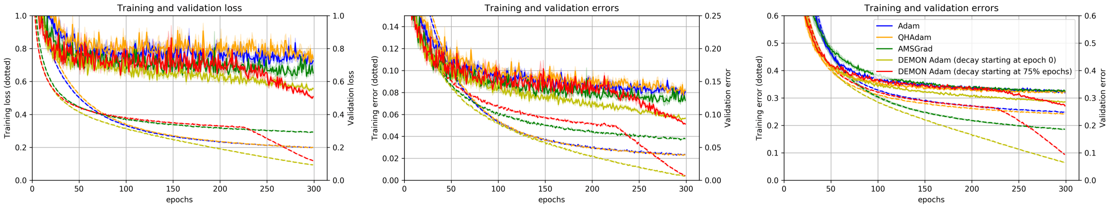

Figure 1: Two left-most plots: `RN18-CIFAR10-DEMONAdam` for 300 epochs. Right-most plot: `VGG16-CIFAR100-DEMONAdam` for 300 epochs. Dotted and solid lines represent training and generalization metrics respectively. Shaded bands represent one standard deviation.

4.2. Adaptive momentum methods

We apply DEMON SGDM (Algorithm 2) to a variety of models and tasks. Since SGDM with learning rate decay is most often used to achieve the state-of-the-art results with the architectures and tasks in question, we include SGDM with learning rate decay as the target to beat. SGDM with

	ResNet18			LSTM		VAE	
	75 epochs	150 epochs	300 epochs	25 epochs	39 epochs	50 epochs	100 epochs
Adam	13.63 ± .22	11.90 ± .06	11.94 ± .06	115.54 ± .64	115.02 ± .52	136.28 ± .18	134.64 ± .14
AMSGrad	13.43 ± .14	11.83 ± .12	10.48 ± .12	108.07 ± .19	107.87 ± .25	137.89 ± .12	135.69 ± .03
QHAdam	15.55 ± .25	13.78 ± .08	13.36 ± .11	112.52 ± .23	112.45 ± .39	136.69 ± .17	134.84 ± .08
Demon Adam	**9.69** ± .10	**8.83** ± .08	**8.44** ± .05	**101.57** ± .32	**101.44** ± .47	**134.46** ± .17	**134.12** ± .08

Table 1: `ResNet18-CIFAR10-DEMONAdam` generalization error, `PTB-LSTM-DEMONAdam` generalization perplexity, `VAE-MNIST-DEMONAdam` generalization loss. The number of epochs was predefined before the execution of the algorithms.

	VGG-16			Wide Residual 16-8			NCSN
	75 epochs	150 epochs	300 epochs	50 epochs	100 epochs	200 epochs	512 epochs
Adam	37.98 ± .20	33.62 ± .11	31.09 ± .09	23.35 ± .20	19.63 ± .26	18.65 ± .07	**8.15** ± .20
AMSGrad	40.67 ± .65	34.46 ± .21	31.62 ± .12	21.73 ± .25	19.35 ± .20	18.21 ± .18	-
QHAdam	36.53 ± .20	32.96 ± .11	30.97 ± .10	21.25 ± .22	19.81 ± .18	18.52 ± .25	-
Demon Adam	**32.40** ± .19	**28.84** ± .18	**27.11** ± .19	**19.42** ± .10	**18.36** ± .11	**17.62** ± .12	8.07 ± .08

Table 2: `VGG16-CIFAR100-DEMONAdam` generalization error, `WideResNet-STL10-DEMONAdam` generalization error, and `NCSN-CIFAR10-DEMONAdam` inception score.

learning rate decay is implemented with a decay on validation error plateau, where we hand-tune the number of epochs to define plateau. Recent adaptive momentum methods included in this section are Aggregated Momentum (AggMo) [22], and Quasi-Hyperbolic Momentum (QHM) [23]. We exclude accelerated SGD [15] due to difficulties in tuning. See Appendix B.2.2 for details. Similar to the last section, we tune all learning rates in roughly multiples of 3 and try to keep all other parameters close to those recommended in the original literature. For Demon SGDM, we leave $\beta_{\text{init}} = 0.9$ for most experiments and generally decay from β_{init} to 0.

	ResNet18		LSTM		VAE	
	75 epochs	300 epochs	25 epochs	39 epochs	50 epochs	100 epochs
SGDM learning rate decay	9.05 ± .07	7.97 ± .14	89.59 ± .07	**87.57** ± .11	140.51 ± .73	139.54 ± .34
AggMo	13.02 ± .23	10.94 ± .12	89.09 ± .16	89.07 ± .15	139.69 ± .17	139.07 ± .26
QHM	12.66 ± .19	10.42 ± .05	94.47 ± .19	94.44 ± .13	145.84 ± .39	140.92 ± .19
Demon SGDM	**8.97** ± .16	**7.58** ± .04	**88.33** ± .16	88.32 ± .12	**139.32** ± .23	**137.51** ± .29

Table 3: `ResNet18-CIFAR10-DEMONSGDM` generalization error, `PTB-LSTM-DEMONSGDM` generalization perplexity, `VAE-MNIST-DEMONSGDM` generalization loss.

	VGG-16			Wide Residual 16-8		
	75 epochs	150 epochs	300 epochs	75 epochs	150 epochs	300 epochs
SGDM learning rate decay	35.29 ± .59	30.65 ± .31	29.74 ± .43	21.05 ± .27	17.83 ± 0.39	15.16 ± .36
AggMo	42.85 ± .89	34.25 ± .24	32.32 ± .18	22.70 ± .11	20.06 ± .31	17.90 ± .13
QHM	42.14 ± .79	33.87 ± .26	32.45 ± .13	22.86 ± .15	19.40 ± .23	17.79 ± .08
Demon SGDM	**34.35** ± .44	**30.59** ± .26	**28.99** ± .16	**19.45** ± .20	**15.98** ± .40	**13.67** ± .13

Table 4: `VGG16-CIFAR100-DEMONSGDM`, `WideResNet-STL10-DEMONSGDM` generalization error.

We report results in tabulated form in Tables 3 and 4. With Demon SGDM, we often achieve better generalization error than SGDM with learning rate decay, a surprising finding given the frequency with which SGDM with learning rate decay achieves state-of-the-art results. Similarly to Demon Adam, Demon SGDM often continues to learn after other methods have plateaued.

References

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

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

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

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

[5] 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.

[6] 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*, pages 1243–1252. JMLR. org, 2017.

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

[8] 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.

[9] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT Press, 2016.

[10] 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*, pages 770–778, 2016.

[11] 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.

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

[13] 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.

[14] 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*, pages 4700–4708, 2017.

[15] 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.

[16] 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)*, pages 1–9. IEEE, 2018.

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

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

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

[20] 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.

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

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

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

[24] 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*, pages 3111–3119, 2013.

[25] 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)*, pages 997–1004. IEEE, 2016.

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

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

[28] 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.

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

[30] 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*, pages 91–99, 2015.

[31] 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.

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

[33] 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)*, pages 4955–4959. IEEE, 2016.

[34] 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.

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

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

[37] 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*, pages 249–256. ACM, 2018.

[38] 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*, pages 1139–1147, 2013.

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

[40] 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.

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

[42] 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*, pages 4148–4158, 2017.

[43] 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*, pages 1492–1500, 2017.

[44] Yuan. On the influence of momentum acceleration on online learning. *Journal of Machine Learning Research*, 17(192):1–66, 2016.

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

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

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

Appendix A. Proof of convergence

We analyze the global convergence of DEMON CM in the convex setting, following [7]. 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 $f(\bar{\theta}_T)$ converges to the optimum f^* with decreasing momentum, where $\bar{\theta}_T$ is the average of θ_t for $t = 1, ..., T$. Our following theorem holds for a constant learning rate and β_t 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:*

$$\beta_t = \tfrac{1}{t} \cdot \tfrac{t+1}{t+2}, \quad \alpha \in \left(0, \tfrac{2}{3L}\right).$$

We consider the CM iteration in non-stochastic settings, where:

$$\theta_{t+1} = \theta_t - \alpha \nabla f(\theta_t) + \beta_t \left(\theta_t - \theta_{t-1}\right).$$

Then, the sequence $\{\theta_t\}_{t=1}^T$ generated by the CM iteration, with decreasing momentum, satisfies:

$$f(\bar{\theta}_T) - f^* \leq \tfrac{\|\theta_1 - \theta^\star\|^2}{T} \left(\tfrac{3}{4}L + \tfrac{1}{2\alpha}\right),$$

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

Proof. Let $\beta_t = \frac{1}{t} \cdot \frac{t+1}{t+2}$ and

$$p_t = \tfrac{1}{t}(\theta_t - \theta_{t-1}).$$

We consider the CM iteration in non-stochastic settings, where:

$$\theta_{t+1} = \theta_t - \alpha \nabla f(\theta_t) + \beta_t \left(\theta_t - \theta_{t-1}\right).$$

Using the definition of p_t above, one can easily prove that:

$$\theta_{t+1} + p_{t+1} = (1 + \frac{1}{t+1})\theta_{t+1} - \frac{1}{t+1}\theta_t = \theta_t + p_t - \tfrac{\alpha(t+2)}{t+1}\nabla f(\theta_t).$$

Using this expression, we will analyze the term $\|\theta_{t+1} + p_{t+1} - \theta^\star\|_2$:

$$\begin{aligned}
\|\theta_{t+1} + p_{t+1} - \theta^\star\|^2 &= \|\theta_t + p_t - \theta^\star\|^2 - \tfrac{2\alpha(t+2)}{t+1}\left\langle \theta_t + p_t - \theta^\star, \nabla f(\theta_t)\right\rangle + \left(\tfrac{\alpha(t+2)}{t+1}\right)^2 \cdot \|\nabla f(\theta_t)\|^2 \\
&= \|\theta_t + p_t - \theta^\star\|^2 - \tfrac{2\alpha(t+2)}{t(t+1)}\left\langle \theta_t - \theta_{t-1}, \nabla f(\theta_t)\right\rangle \\
&\quad - \tfrac{2\alpha(t+2)}{t+1}\left\langle \theta_t - \theta^\star, \nabla f(\theta_t)\right\rangle + \left(\tfrac{\alpha(t+2)}{t+1}\right)^2 \cdot \|\nabla f(\theta_t)\|^2
\end{aligned}$$

Since f is convex, continuously differentiable, its gradient is Lipschitz continuous with constant L, then

$$\tfrac{1}{L}\|\nabla f(\theta_t)\|^2 \leq \langle \theta_t - \theta^\star, \nabla f(\theta_t)\rangle, \tag{2}$$

$$f(\theta_t) - f^* + \tfrac{1}{2L}\|\nabla f(\theta_t)\|^2 \leq \langle \theta_t - \theta^\star, \nabla f(\theta_t)\rangle, \tag{3}$$

$$f(\theta_t) - f(\theta_{t-1}) \leq \langle \theta_t - \theta_{t-1}, \nabla f(\theta_t)\rangle. \tag{4}$$

Substituting the above inequalities leads to

$$\begin{aligned}\|\theta_{t+1}+p_{t+1}-\theta^\star\|^2 \leq \|\theta_t+p_t-\theta^\star\|^2 &- \tfrac{2\alpha(t+2)}{t(t+1)}(f(\theta_t)-f(\theta_{t-1})) \\ &- 2\alpha\tfrac{(1-\lambda)(t+2)}{L(t+1)}\cdot\|\nabla f(\theta_t)\|^2 - 2\alpha\lambda\tfrac{t+2}{t+1}(f(\theta_t)-f^*) \\ &- \left(\alpha\tfrac{\lambda(t+2)}{L(t+1)}\right)\cdot\|\nabla f(\theta_t)\|^2 + \left(\frac{\alpha(t+2)}{t+1}\right)^2\cdot\|\nabla f(\theta_t)\|^2\end{aligned}$$

where $\lambda \in (0,1]$ is a parameter weighting (2) and (3). Grouping together terms yields

$$\begin{aligned}\left(\tfrac{2\alpha(t+2)}{t(t+1)}+\tfrac{2\alpha\lambda(t+2)}{t+1}\right)(f(\theta_t)-f^*) &+ \|\theta_{t+1}+p_{t+1}-\theta^\star\|^2 \\ &\leq \tfrac{2\alpha(t+2)}{t(t+1)}(f(\theta_{t-1})-f^*) + \|\theta_t+p_t-\theta^\star\|^2 \\ &\quad + \tfrac{\alpha(t+2)}{t+1}\left(\tfrac{\alpha(t+2)}{t+1}-\tfrac{2(1-\lambda)}{L}-\tfrac{\lambda}{L}\right)\|\nabla f(\theta_t)\|^2.\end{aligned}$$

The last term is non-positive when $\alpha \in [0, \frac{t+1}{t+2}(\frac{2-\lambda}{L})]$ so it can be dropped. Summing over $t = 1, ..., T$ yields

$$\begin{aligned}2\alpha\lambda\sum_{t=1}^{T}\tfrac{t+2}{t+1}(f(\theta_t)-f^*) &+ \sum_{t=1}^{T}\left(\tfrac{2\alpha(t+2)}{t(t+1)}(f(\theta_t)-f^*)+\|\theta_{t+1}+p_{t+1}-\theta^\star\|^2\right) \\ &\leq \sum_{t=1}^{T}\left(\tfrac{2\alpha(t+2)}{t(t+1)}(f(\theta_{t-1})-f^*)+\|\theta_t+p_t-\theta^\star\|^2\right),\end{aligned}$$

implying that:

$$2\alpha\lambda\sum_{t=1}^{T}\tfrac{t+2}{t+1}(f(\theta_t)-f^*) \leq 3\alpha(f(\theta_1)-f^*)+\|\theta_1-\theta^\star\|^2.$$

Since:

$$2\alpha\lambda\sum_{t=1}^{T}(f(\theta_t)-f^*) \leq 2\alpha\lambda\sum_{t=1}^{T}\tfrac{t+2}{t+1}(f(\theta_t)-f^*) \leq 3\alpha\lambda\sum_{t=1}^{T}(f(\theta_t)-f^*),$$

we further have:

$$3\alpha\lambda\sum_{t=1}^{T}(f(\theta_t)-f^*) \leq \tfrac{3}{2}\Big(3\alpha(f(\theta_1)-f^*)+\|\theta_1-\theta^\star\|^2\Big).$$

Due to the convexity of f,

$$f(\bar{\theta}_t) \leq \tfrac{1}{T}\sum_{t=1}^{T}f(\theta_t),$$

observe that

$$f(\bar{\theta}_T)-f^* \leq \tfrac{1}{T}\sum_{t=1}^{T}(f(\theta_t)-f^*) \leq \tfrac{1}{3\alpha\lambda T}\left(\tfrac{9}{2}\alpha(f(\theta_1)-f^*)+\tfrac{3}{2}\|\theta_1-\theta^\star\|^2\right).$$

Since $f(\theta_1)-f^* \leq \frac{L}{2}\|\theta_1-\theta^\star\|^2$ by Lipschitz continuous gradients, setting $\lambda = 1$ and observing $(t+1)/(t+2) \geq 2/3$ gives the result.

Experiment short name	Model	Dataset	Optimizer
RN18-CIFAR10-DEMONSGDM	ResNet18	CIFAR10	DEMON SGDM
RN18-CIFAR10-DEMONAdam	ResNet18	CIFAR10	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

Table 5: Summary of experimental settings.

Appendix B. Experiments

We evaluated the momentum decay rule with Adam and SGDM on Residual CNNs, Non Residual CNNS, RNNs and generative models. For CNNs, we used the image classification datasets CIFAR10, CIFAR100 and STL10 datasets. For RNNs, we used the language modeling dataset PTB. For generative modeling, we used the MNIST and CIFAR10 datasets. For each network dataset pair other than NSCN, we evaluated Adam, QHAdam, AMSGrad, DEMON Adam, AggMo, QHM, DEMON SGDM, and SGDM with learning rate decay. For adaptive learning rate methods and adaptive momentum methods, we generally perform a grid search over the learning rate. For SGDM, we generally perform a grid search over learning rate and initial momentum. For SGDM learning rate decay, the learning rate is decayed by a factor of 0.1 after there is no improvement in validation loss for the best of $\{1, 2, 3, 5, 10, 20, 30, 40\}$ epochs.

B.1. Setup

We describe the six test problems in this paper.

- **CIFAR10 - ResNet18** CIFAR10 contains 60,000 32x32x3 images with a 50,000 training set, 10,000 test set split. There are 10 classes. ResNet18 [10] is an 18 layers 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 [35] 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 [45] 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 [12] with 2 layers, 650 units per layer, and dropout of 0.5. Trained with a batch size of 20.

- **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 [18] 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 [36] 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. We follow the exact implementation in and defer details to the original paper.

B.2. Methods

B.2.1. Adaptive learning rate

Adam [17], as previously introduced in section **??**, keeps an exponentially decaying average of squares of past gradients to adapt the learning rate. It also introduces an exponentially decaying average of gradients.

The Adam algorithm is parameterized by learning rate $\eta > 0$, discount factors $\beta_1 < 1$ and $\beta_2 < 1$, a small constant ϵ, and uses the update rule:

$$\begin{aligned} \mathcal{E}^{g}_{t+1} &= \beta_1 \cdot \mathcal{E}^{g}_{t} + (1-\beta_1)\cdot g_t, \\ \mathcal{E}^{g\circ g}_{t+1} &= \beta_2 \cdot \mathcal{E}^{g\circ g}_{t} + (1-\beta_2)\cdot (g_t \circ g_t), \\ \theta_{t+1,i} &= \theta_{t,i} - \frac{\eta}{\sqrt{\mathcal{E}^{g\circ g}_{t+1,i} + \varepsilon}} \cdot \mathcal{E}^{g}_{t+1,i}, \quad \forall t. \end{aligned}$$

AMSGrad [29] resolves an issue in the proof of Adam related to the exponential moving average $\mathcal{E}^{g\circ g}_{t}$, where Adam does not converge for a simple optimization problem. Instead of an exponential moving average, AMSGrad keeps a running maximum of $\mathcal{E}^{g\circ g}$.

The AMSGrad algorithm is parameterized by learning rate $\eta > 0$, discount factors $\beta_1 < 1$ and $\beta_2 < 1$, a small constant ϵ, and uses the update rule:

$$\begin{aligned} \mathcal{E}^{g}_{t+1} &= \beta_1 \cdot \mathcal{E}^{g}_{t} + (1-\beta_1)\cdot g_t, \\ \mathcal{E}^{g\circ g}_{t+1} &= \beta_2 \cdot \mathcal{E}^{g\circ g}_{t} + (1-\beta_2)\cdot (g_t \circ g_t), \\ \hat{\mathcal{E}}^{g\circ g}_{t+1,i} &= max(\hat{\mathcal{E}}^{g\circ g}_{t,i}, \mathcal{E}^{g\circ g}_{t,i}), \\ \theta_{t+1,i} &= \theta_{t,i} - \frac{\eta}{\sqrt{\hat{\mathcal{E}}^{g\circ g}_{t+1,i} + \varepsilon}} \cdot \mathcal{E}^{g}_{t+1,i}, \quad \forall t, \end{aligned}$$

where $\mathcal{E}^{g}_{t+1}$ and $\mathcal{E}^{g\circ g}_{t+1}$ are defined identically to Adam.

QHAdam (Quasi-Hyperbolic Adam) [23] extends QHM (Quasi-Hyperbolic Momentum), introduced further below, to replace both momentum estimators in Adam with quasi-hyperbolic terms. This quasi-hyperbolic formulation is capable of recovering Adam and NAdam [4], amongst others.

The QHAdam algorithm is parameterized by learning rate $\eta > 0$, discount factors $\beta_1 < 1$ and $\beta_2 < 1$, $\nu_1, \nu_2 \in \mathbb{R}$, a small constant ϵ, and uses the update rule:

$$
\begin{aligned}
\mathcal{E}^{g}_{t+1} &= \beta_1 \cdot \mathcal{E}^{g}_{t} + (1-\beta_1)\cdot g_t,\\
\mathcal{E}^{g\circ g}_{t+1} &= \beta_2 \cdot \mathcal{E}^{g\circ g}_{t} + (1-\beta_2)\cdot (g_t \circ g_t),\\
\hat{\mathcal{E}}^{g}_{t+1} &= (1+\beta_1^{t+1})^{-1}\cdot \mathcal{E}^{g}_{t+1},\\
\hat{\mathcal{E}}^{g\circ g}_{t+1} &= (1+\beta_2^{t+1})^{-1}\cdot \mathcal{E}^{g\circ g}_{t+1},\\
\theta_{t+1,i} &= \theta_{t,i} - \eta\left[\frac{(1-\nu_1)\cdot g_t + \nu_1\cdot \hat{\mathcal{E}}^{g}_{t+1}}{\sqrt{(1-\nu_2)g_t^2 + \nu_2\cdot \hat{\mathcal{E}}^{g\circ g}_{t+1}} + \epsilon}\right], \quad \forall t,
\end{aligned}
$$

where $\mathcal{E}^{g}_{t+1}$ and $\mathcal{E}^{g\circ g}_{t+1}$ are defined identically to Adam.

B.2.2. ADAPTIVE MOMENTUM

AggMo (Aggregated Momentum) [22] takes a linear combination of multiple momentum buffers. It maintains K momentum buffers, each with a different discount factor, and averages them for the update.

The AggMo algorithm is parameterized by learning rate $\eta > 0$, discount factors $\beta \in \mathbb{R}^K$, and uses the update rule:

$$
\begin{aligned}
(\mathcal{E}^{g}_{t+1})^{(i)} &= \beta^{(i)}\cdot (\mathcal{E}^{g}_{t})^{(i)} + g_t, \quad \forall i \in [1, K],\\
\theta_{t+1,i} &= \theta_{t,i} - \eta\left[\frac{1}{K}\cdot \sum_{i=1}^{K} (\mathcal{E}^{g}_{t+1})^{(i)}\right], \quad \forall t.
\end{aligned}
$$

QHM (Quasi-Hyperbolic Momentum) [23] is a weighted average of the momentum and plain SGD. QHM is capable of recovering Nesterov Momentum [26], Synthesized Nesterov Variants [20], accSGD [15] and others.

The QHM algorithm is parameterized by learning rate $\eta > 0$, discount factor $\beta < 1$, immediate discount factor $\nu \in \mathbb{R}$, and uses the update rule:

$$
\begin{aligned}
\mathcal{E}^{g}_{t+1} &= \beta\cdot \mathcal{E}^{g}_{t} + (1-\beta)\cdot g_t,\\
\theta_{t+1,i} &= \theta_{t,i} - \eta\Big[(1-\nu)\cdot g_t + \nu\cdot \mathcal{E}^{g}_{t+1}\Big], \quad \forall t.
\end{aligned}
$$

B.3. Additional results for adaptive learning rate methods

In Figure 2, we display randomly selected images from the CIFAR10 dataset, either real, generated by NCSN trained with Adam, and generated by NCSN trained with DEMON Adam. Images produced by NCSN trained with Adam appear noticeably bright green.

B.4. Additional results for adaptive momentum methods

Learning curves for various datasets are given in Figure 3

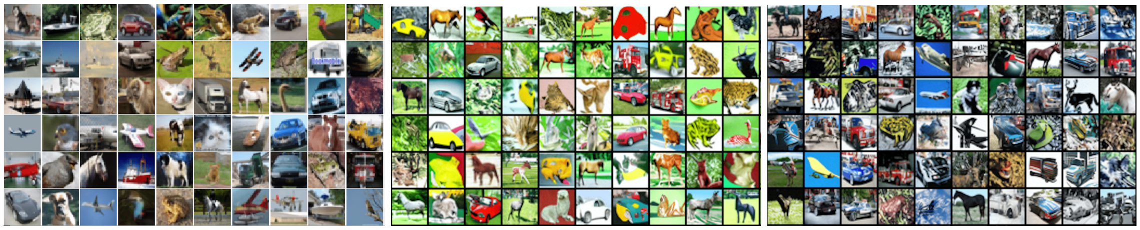

Figure 2: Randomly selected CIFAR10 images generated with NCSN. Left: Real CIFAR10 images. Middle: Adam. Right: DEMON Adam.

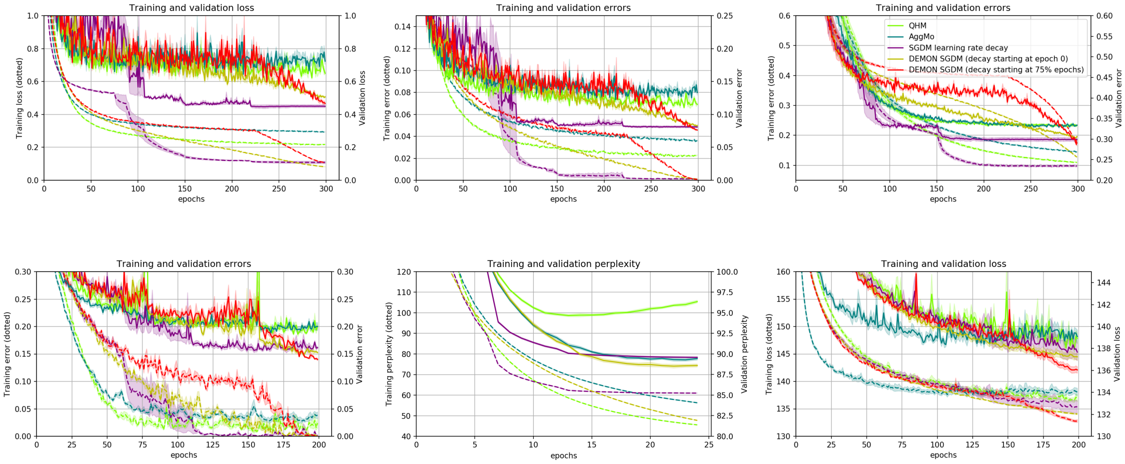

Figure 3: Top row, two left-most plot: `RN18-CIFAR10-DEMONSGDM` for 300 epochs. Top row, right-most plot: `VGG16-CIFAR100-DEMONSGDM` for 300 epochs. Bottom row, left-most plots: `WRN-STL10-DEMONSGDM` for 200 epochs. Bottom row, middle plot: `PTB-LSTM-DEMONSGDM` for 25 epochs. Bottom row, right-most plot: `VAE-MNIST-DEMONSGDM` for 200 epochs. Dotted and solid lines represent training and generalization metrics respectively. Shaded bands represent 1 standard deviation.