The Marginal Value of Adaptive Gradient Methods in Machine Learning

Jian Yuan, Yixu Gao
19210980107, 19210980111
School of Data Science
Fudan University

Abstract

Adaptive optimization methods such as AdaGrad, RMSProp and Adam are getting increasingly popular in training deep neural networks in recent years. These optimization methods aim to construct a metric from previous iterates and perform local optimization in a more rapid way. The paper shows that for simple, overparameterized problems, adaptive optimization methods can find solutions following some latent pattern other than solutions that gradient descent (GD) or stochastic gradient descent (SGD) find. We follow the specific paper and study the empirical generalization capability of adaptive methods both theoretically by deriving how adaptive methods come to "wrong" solutions, and practically on several state-of-the-art deep learning models. We observe that the solutions found by adaptive methods generalize worse (often significantly worse) than SGD, even when these solutions have better training performance. These results suggest that when training a deep neural network, it is critical to rethink whether adaptive methods should be used.

Introduction

An increasing number of deep learning researchers are using *adaptive gradient methods* to train their models due to their rapid training efficiency[6]. *Adam*, particularly has almost become the default algorithm for deep learning frameworks. However, the generalization and out-of-sample behavior of such adaptive gradient methods remains both poorly understood and poorly performing. Note that in practice, deep learning models follow the pattern that finally minimizes the objective on training set, it is natural to suppose that different optimization algorithm can influence the final model we get, which is also in line with our practice.

Extremely, when the dimensions (i.e. number of parameters) exceeds the number of our data points, it is highly likely that the optimization algorithm matters which final model we obtain. Then comes the question, how can we tell which optimization algorithm can give final model that can generalize better? In this project, we first introduce different non-adaptive and adaptive optimization algorithms, then show that adaptive and non-adaptive optimization methods can indeed find very different solutions with very different generalization properties. We provide a simple generative model for binary classification where the population is linearly separable (i.e., there exists a solution with large margin), but AdaGrad, RMSProp, and Adam converge to a solution that incorrectly classifies new data with probability arbitrarily close to half. On this same example, SGD finds a solution with zero error on new data. Our construction suggests that adaptive methods tend to give undue influence to spurious features that have no effect on out-of-sample generalization.

Additionally, we present practical numerical experiments on deep neural network task, demonstrating that adaptive methods generalize worse than their non-adaptive counterparts. Our experiments reveal three primary findings. First, with the same amount of hyperparameter tuning, SGD and SGD with momentum outperform adaptive methods on the development/test set across all evaluated models and tasks. This is true even when the adaptive methods achieve the same training loss or lower than non-adaptive methods. Second, adaptive methods often display faster initial progress on the training set, but their performance quickly plateaus on the development/test set. Third, the same amount of tuning was required for all methods,

including adaptive methods. This challenges the conventional wisdom that adaptive methods require less tuning. Therefore, it deserves rethinking when deep neural network researchers are wondering which sort of optimization algorithm to use since the adaptive ones and non-adaptive ones can converge to very different results.

Background

Non-adaptive optimization algorithms

The canonical optimization algorithms used to minimize risk are either stochastic gradient methods or stochastic momentum methods.

Stochastic Gradient Descent Stochastic gradient descent (often abbreviated SGD) is an iterative method for optimizing an objective function with suitable smoothness properties (e.g. differentiable or subdifferentiable). It can be regarded as a stochastic approximation of gradient descent optimization, since it replaces the actual gradient (calculated from the entire data set) by an estimate thereof (calculated from a randomly selected subset of the data). Especially in big data applications this reduces the computational burden, achieving faster iterations in trade for a slightly lower convergence rate. It can generally be written as:

$$w_{k+1} = w_k - \alpha\tilde{\nabla} f(w_k).$$

where $\tilde{\nabla} f(w_k) := \nabla f(w_k; x_{ik})$ is the gradient of some loss function f computed on a batch of data x_{ik}.

Stochastic Momentum Descent Further proposals include the momentum method, which appeared in Rumelhart, Hinton and Williams' seminal paper on backpropagation learning. Stochastic momentum methods are the second family of techniques that have been used to accelerate training. These methods can generally be written as:

$$w_{k+1} = w_k - \alpha_k\tilde{\nabla} f(w_k + \gamma_k(w_k - w_{k-1})) + \beta_k(w_k - w_{k-1}).$$

Two popular stochastic momentum descent methods are Polyak's heavy-ball method (HB) and Nesterov's Accelerated Gradient method (NAG), mainly differ in the value of γ_k's:

- **HB:** $\gamma_k = 0$
- **NAG:** $\gamma_k = \beta_k$

Adaptive optimization algorithms

Adaptive gradient and adaptive momentum methods (including AdaGrad, RMSProp, and Adam), other than previous methods, choose a local distance measure constructed using the entire sequence of iterates $(w1, ..., wk)$.

AdaGrad (by Duchi, Hazan, and Singer, 2011) *AdaGrad* (for adaptive gradient algorithm) is a modified stochastic gradient descent algorithm with per-parameter learning rate, first published in 2011. Informally, this increases the learning rate for sparser parameters and decreases the learning rate for ones that are less sparse. This strategy often improves convergence performance over standard stochastic gradient descent in settings where data is sparse and sparse parameters are more informative. Examples of such applications include natural language processing and image recognition. It still has a base learning rate α, but this is multiplied with the elements of a vector $H_{j,j}$ which is the diagonal of the outer product matrix.

$$w_{k+1} := w_k - \alpha H_k^{-1}\tilde{\nabla} f(w_k)$$

$$G_{k+1} := G_k + D_k$$

$$(H = \sqrt{G})$$

Where D_k is a diagonal matrix (with $\tilde{\nabla} f(w_k)$ squared for each entry):

$$D_k := \text{Diag}\{\tilde{\nabla} f(w_k)^2\}$$

So G is linear combinations of the square of past gradient components.

Note:

Different learning rate for each parameter. Larger updates for "infrequent" parameters (small square of gradient over updates) and small updates for "frequent" parameters (large square of gradient over updates).

This feature of adaptive gradient methods is what can lead it give undue influence to "spurious" features.

RMSProp (by Hinton, unpublished) *RMSProp* (for Root Mean Square Propagation) is also a method in which the learning rate is adapted for each of the parameters. The idea is to divide the learning rate for a weight by a running average of the magnitudes of recent gradients for that weight. So, first the running average is calculated in terms of means square.The method is highly similar to AdaGrad, but with different weighting scheme:

$$w_{k+1} := w_k - \alpha H_k^{-1} \tilde{\nabla} f(w_k)$$
$$G_{k+1} := \beta_2 G_k + (1-\beta_2) D_k$$
$$(H = \sqrt{G})$$

ADAM (by Kingma and Ba, 2014) *Adam* (short for Adaptive Moment Estimation) is an update to the RMSProp optimizer. In this optimization algorithm, running averages of both the gradients and the second moments of the gradients are used. Adam's parameter update is given by:

$$w_{k+1} := w_k - \alpha \hat{H}_{k+1}^{-1} \hat{g}_{k+1}$$
$$g_{k+1} := \beta_1 g_k + (1-\beta_1) \tilde{\nabla} f(w_k)$$
$$G_{k+1} := \beta_2 G_k + (1-\beta_2) D_k$$
$$\hat{g}_{k+1} = \frac{g_{k+1}}{1-\beta_1^{k+1}}$$
$$\hat{G}_{k+1} = \frac{G_{k+1}}{1-\beta_2^{k+1}}$$
$$\left(\hat{H} = \sqrt{\hat{G}}\right)$$

(Note in β_1^k, k denotes β_2 to the power k.)

Adaptive methods attempt to adjust an algorithm to the geometry of the data. In contrast, stochastic gradient descent and related variants use the l_2 geometry inherent to the parameter space, and are equivalent to setting $H_k = I$ in the adaptive methods.

In this context, generalization refers to the performance of a solution w on a broader population. Performance is often defined in terms of a different loss function than the function f used in training. For example, in classification tasks, we typically define generalization in terms of classification error rather than cross-entropy.

Simple Setting for Analysis: Gradient Descent for Linear Regression

In order to prove theoretically that adaptive optimization algorithms and non-adaptive ones can produce solutions with very different properties, let's consider such an extreme case:

$$\min_w R[w] := ||Xw - y||^2$$

with

- X is a $n \times d$ matrix (n for data points; d for features)
- y is a n-dimensional vector of labels in $\{-1, 1\}$

We assume the problem is over-parameterized / over-determined, so $d > n$ (number of features ¿ number of data points).

Thus if there is a w that achieves loss 0, then there are infinitely many global minimizers (solution not unique, since over-complete).

Calculating the Gradient

$$R = (Xw - y)^T (Xw - y) =$$
$$(Xw)^T (Xw - y) - y^T (Xw - y) =$$
$$(Xw)^T (Xw) - (Xw)^T y - y^T (Xw) + y^T y$$
$$w^T X^T X w - 2 y^T X w + y^T y$$
$$w^T X^T X w - 2 (X^T y)^T w + y^T y$$

Using

$$\frac{d}{dw} y^T w = \frac{d}{dw} w^T y = y$$
$$\frac{d}{dw} w^T X w = (X + X^T) w$$

We get:

$$\frac{d}{dw} R = (X^T X + X X^T) w - 2 X^T y = 2 X^T X w - 2 X^T y$$

Thus, $\frac{d}{dw} R \in$ span$\{$ rows of X $\}$, since for arbitrary vector a, $X^T a$ for is a linear combination of the rows of X.

Non-adaptive methods

Claim: If w is initialized in the row span of X (e.g. $w = 0$), and uses only linear combination of gradients / stochastic gradients, w_t must also lie in the row span of X, for all updates t.

Justification: If we start in the span of the rows of X, and add linear combinations of the rows of X, we will still end up in the span of X's rows.

Claim: The unique solution that lies in the row span of X also happens to be the solution with **minimum Euclidean norm**, so

$$w^{SGD} = X^T (X X^T)^{-1} y$$

Justification: The idea is that since the system is underdetermined, we can think of the problem of finding vector as w, as finding the vector in a very high dimensional space, that projects to another vector y in a lower dimensional space (the projection function is determined by X).

$$\min_w R[w] := ||Xw - y||^2$$

Here we come to an important idea – the w with the minimum norm in this case generalizes the best.

Adaptive methods

Claim: Suppose $X^T y$ has no components equal to 0, and there exists a scalar c such that $Xsign(X^T y) = cy$.

Then when initialized at $w_0 = 0$, AdaGrad, Adam, and RMSProp all converge to the solution $w \propto sign(X^T y)$.

Justification: We want to show for all iterations k,

$$w_k = \lambda_k sign(X^T y)$$

We prove by induction.

Base case We know the assertion holds for $w_0 = 0$, as $\lambda_0 = 0$.

Inductive step Now, given that for some λ_{k-1},

$$\nabla R(w_{k-1}) = \lambda_{k-1} sign(X^T y)$$

We can show that for some μ_k, (details in paper)

$$\nabla R(w_k) = \mu_k X^T y$$

The problematic part - what differentiates the adaptive gradient methods - is the H_k term. Letting $g_k = \nabla R(w_k)$, then H_k is diagonal matrix such that

$$H_k = diag\left\{\sqrt{g_k^2}\right\} = diag\left\{\sqrt{\mu_k}|X^T y|\right\} =: v_k diag(|X^T y|)$$

$$\begin{aligned} w_{k+1} &= w_k - \alpha_k H_k^{-1} \tilde{\nabla} f(w_k + \gamma_k(w_k - w_{k-1})) + \\ &\quad \beta_k H_k^{-1} H_{k-1}(w_k - w_{k-1}) \\ &= \lambda_k sign(X^T y) - \alpha_k (v_k |X^T y|)^{-1} \mu_k X^T y + \\ &\quad \beta_k (v_k |X^T y|)^{-1} v_{k-1} |X^T y| (\lambda_k - \lambda_{k-1}) sign(X^T y) \\ &= \left\{\lambda_k - \frac{\alpha_k \mu_k}{v_k} + \frac{\beta_k v_{k-1}}{v_k}(\lambda_k - \lambda_{k-1})\right\} sign(X^T y) \end{aligned}$$

Adaptivity Can Overfit

Problem Setting Infinite dimensional, n examples. Labels $y^{(i)} \in \{-1, 1\}$, assigned 1 with probability $p > \frac{1}{2}$.

$$x_j^{(i)} = \begin{cases} y^{(i)}, & j = 1, \\ 1, & j = 2, 3, \\ 1, & j = 4 + 5(i-1), ..., 4 + 5(i-1) + 2(1 - y^{(i)}) \\ 0, & otherwise \end{cases}$$

- Only the first feature is useful (class label)
- Next two features are always 1
- All other features are unique for each n

  If there label is 1, 1 unique features

  If there label is -1, 5 unique features

SGD Solution SGD will find minimum norm solution –¿ no generalization problem.

Let P and N by the set of positive and negative examples respectively. α_+ and α_- are non-negative constants (can be solved for in closed form; see paper).

$$w^{sgd} = \sum_{i \in P} \alpha_+ x_i - \sum_{j \in N} \alpha_- x_j$$

AdaGrad Solution

$$u = X^T y = w$$

$$b = \sum_{i=1}^{n} y^{(-i)}$$

$$u_j = \begin{cases} n, & j = 1, \\ b, & j = 2, 3, \\ y_j, & \text{if } j > 3 \text{ and } x_j = 1 \\ 0, & \text{otherwise} \end{cases}$$

Satisfies conditions of lemma since $< u, x_i >= y_i + 2 + y_i(3 - 2y_i) = 4y_i$; so

$$w^{ada} \propto \text{sign}(u)$$

So all components of w^{ada} either zero or $\pm\tau$ for some constant τ.

For a new point though, x^{test}, the only features that are nonzero for both x^{test} and w^{ada} are the first three. Thus, the solution will label all unseen data as being in the positive class:

$$< w^{ada}, x^{test} >= \tau(y^{test} + 2) > 0$$

Experiment & Analysis

From previous discussion, we have come to the conclusion that different optimization algorithms can converge to very different solutions, we now turn to an empirical study of deep neural networks to see whether we observe a similar discrepancy in generalization. We compare two non-adaptive methods – SGD and the heavy ball method (HB) – to three popular adaptive methods – AdaGrad, RMSProp and Adam. We study performance on a deep learning problem: discriminative parsing on Penn Treebank.

We conduct the experiment 5 times from randomly initialized starting points, using the initialization scheme specified in each code repository. We allocate a pre-specified budget on the number of epochs used for training each model. When a development set was available, we chose the settings that achieved the best peak performance on the development set by the end of the fixed epoch budget. The detailed experiment settings are shown in Figure 1.

Our experiments show the following primary findings:

- Adaptive methods find solutions that generalize worse than those found by non-adaptive methods.
- Even when the adaptive methods achieve the same training loss or lower than non-adaptive methods, the development or test performance is worse.
- Adaptive methods often display faster initial progress on the training set, but their performance quickly plateaus on the development set.
- Though conventional wisdom suggests that Adam does not require tuning, we find that tuning the initial learning rate and decay scheme for Adam yields significant improvements over its default settings in all cases.

Training Objective and Models

Constituency Parsing A constituency parser is used to predict the hierarchical structure of a sentence, breaking it down into nested clause-level, phrase-level, and word-level units. We carry out experiments using two stateof-the-art parsers: the stand-alone discriminative parser of Cross and Huang , and the generative reranking parser of Choe and Charniak. In both cases, we use the dev-decay scheme with = 0.9 for learning rate decay. Cross and Huang develop a transition-based framework that reduces constituency parsing to a sequence prediction problem, giving a one-to-one correspondence between parse trees and sequences of structural and labeling actions. Using their code with the default settings, we trained for 50 epochs on the Penn Treebank , comparing labeled F1 scores on the training and development data over time. Note that RMSProp was not implemented in the used version of DyNet when the original paper was published, but in our project we managed to implement this algorithm and received its performance. Results are shown in Figure 3 and Figure **??**. It can be found that SGD obtained the best overall performance on the development set, followed closely by HB and Adam, with AdaGrad falling far behind. The default configuration of Adam without learning rate decay actually achieved the best overall training performance by the end of the run, but was notably worse than tuned Adam on the development set. Interestingly, Adam achieved its best development F1 of 91.11 quite early, after just 6 epochs, whereas SGD took 18 epochs to reach this value and didn't reach its best F1 of 91.24 until epoch 31. On the other hand, Adam continued to improve on the training set well after its best development performance was obtained, while the peaks for SGD were more closely aligned. Often adaptive gradient methods outperform SGD early on in training, but their test error of AdaGrad's rate of improvement flatlines earliest.

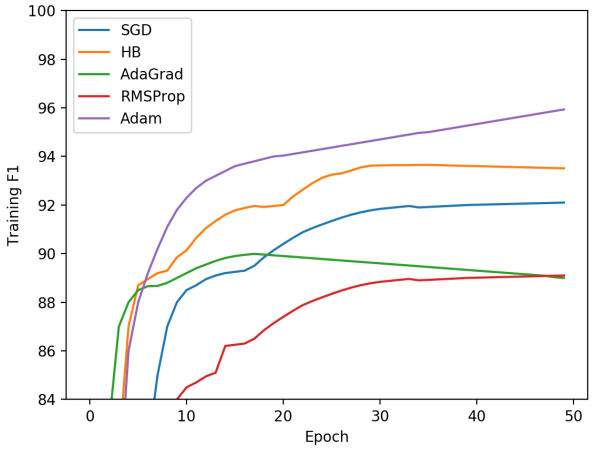

Figure 2: Discriminative Parsing (Training Set)

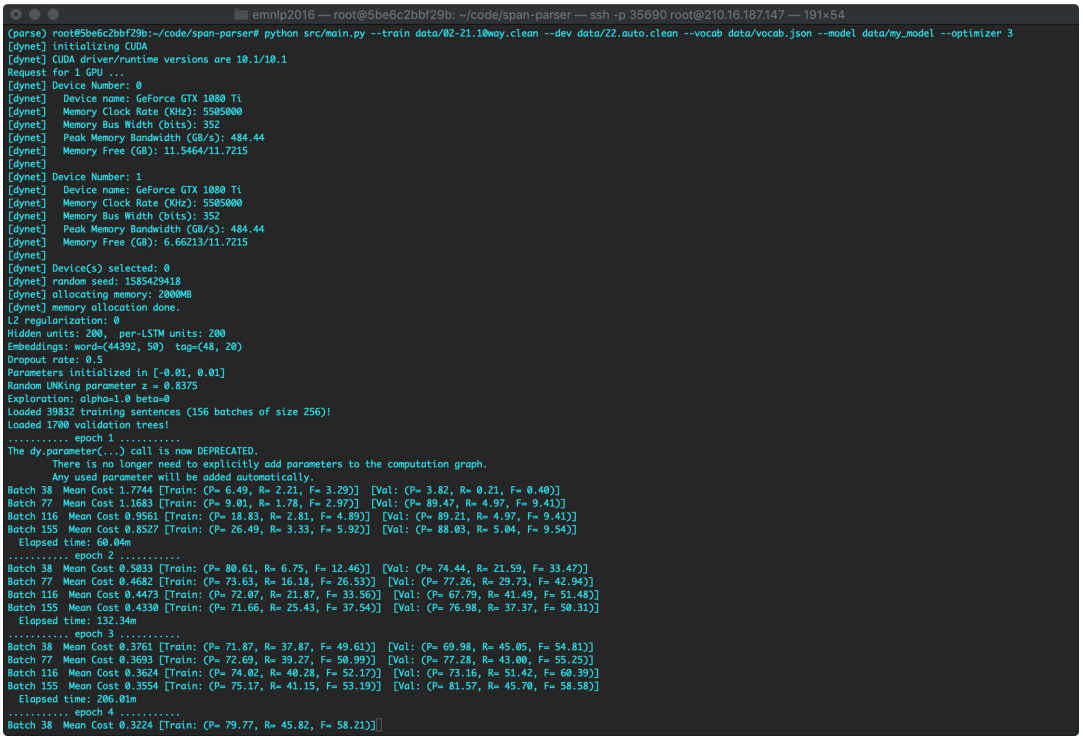

Figure 1: Discriminative Parsing (Training Set)

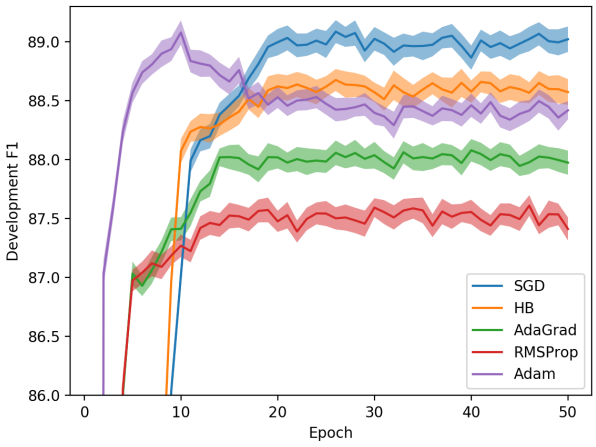

Figure 3: Discriminative Parsing (Development Set)

Conclusion

In our project, we have the following findings:

- In over-parameterized settings, using adaptive gradient methods can give undue influence to spurious features and lead to minima that lead to worse generalization.
- The difference in generalization (empirically) using SGD vs. Adaptive gradient methods is sometimes little, but still significant.

- Exploring what generalization and regularization in the context of deep learning and over-parameterized settings is very interesting.

Despite the fact that our experimental evidence demonstrates that adaptive methods are not always advantageous for machine learning, the Adam algorithm remains incredibly popular. Still, adaptive gradient methods are suitable = for training GANs and Q-learning with function approximation since both of these applications are not solving optimization problems. It is possible that the dynamics of Adam are accidentally well matched to these sorts of optimization-free iterative search procedures. It is also possible that carefully tuned stochastic gradient methods may work as well or better in both of these applications. Therefore, deep neural network researchers had better rethink which optimization algorithm to use before implementation.

References

[1] Wilson, Ashia C., et al. "The marginal value of adaptive gradient methods in machine learning." Advances in Neural Information Processing Systems. 2017.

[2] Cross, James, and Liang Huang. "Span-based constituency parsing with a structure-label system and provably optimal dynamic oracles." arXiv preprint arXiv:1612.06475 (2016).

[3] John C. Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. Journal of Machine Learning Research, 12:2121–2159, 2011.

[4] Sepp Hochreiter and Jürgen Schmidhuber. Flat minima. Neural Computation, 9(1):1–42, 1997.

[5] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. arXiv:1611.07004, 2016.

[6] Andrej Karparthy. A peek at trends in machine learning. https://medium.com/@karpathy/a-peek-at-trends-in-machine-learning-ab8a1085a106. Accessed: 2017-05-17.

[7] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima. In The International Conference on Learning Representations (ICLR), 2017.

[8] D.P. Kingma and J. Ba. Adam: A method for stochastic optimization. The International Conference on Learning Representations (ICLR), 2015.

[9] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. In International Conference on Learning Representations (ICLR), 2016.

[10] Siyuan Ma and Mikhail Belkin. Diving into the shallows: a computational perspective on large-scale shallow learning. arXiv:1703.10622, 2017.

[11] Mitchell P. Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. Building a large annotated corpus of english: The penn treebank. COMPUTATIONAL LINGUISTICS, 19(2):313–330, 1993.

[12] H. Brendan McMahan and Matthew Streeter. Adaptive bound optimization for online convex optimization. In Proceedings of the 23rd Annual Conference on Learning Theory (COLT), 2010.

[13] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In International Conference on Machine Learning (ICML), 2016.

[14] Behnam Neyshabur, Ruslan Salakhutdinov, and Nathan Srebro. Path-SGD: Path-normalized optimization in deep neural networks. In Neural Information Processing Systems (NIPS), 2015.

[15] Behnam Neyshabur, Ryota Tomioka, and Nathan Srebro. In search of the real inductive bias: On the role of implicit regularization in deep learning. In International Conference on Learning Representations (ICLR), 2015.

[16] Maxim Raginsky, Alexander Rakhlin, and Matus Telgarsky. Non-convex learning via stochastic gradient Langevin dynamics: a nonasymptotic analysis. arXiv:1702.03849, 2017.

[17] Benjamin Recht, Moritz Hardt, and Yoram Singer. Train faster, generalize better: Stability of stochastic gradient descent. In Proceedings of the International Conference on Machine Learning (ICML), 2016.

[18] Scott Reed, Zeynep Akata, Xinchen Yan, Lajanugen Logeswaran, Bernt Schiele, and Honglak Lee. Generative adversarial text to image synthesis. In Proceedings of The International Conference on Machine Learning (ICML), 2016.

[19] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In Proceedings of the International Conference on Machine Learning (ICML), 2013.

[20] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016.

[21] T. Tieleman and G. Hinton. Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning, 2012.

[22] Yuan Yao, Lorenzo Rosasco, and Andrea Caponnetto. On early stopping in gradient descent learning. Construc-

tive Approximation, 26(2):289–315, 2007.
[23] Sergey Zagoruyko. Torch blog. http://torch.ch/blog/2015/07/30/cifar.html, 2015.
[24] Charniak, Eugene. ”Parsing as language modeling.” Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing. 2016.