International Journal of Internet, Broadcasting and Communication Vol.11 No.4 31-36 (2019)
http://dx.doi.org/10.7236/IJIBC.2019.11.4.31

IJIBC 19-4-5

FAST-ADAM in Semi-Supervised Generative Adversarial Networks

Li Kun, Dae-Ki Kang

Department of Computer Engineering, Dongseo University, Busan, Korea
likun570117572@gmail.com, dkkang@dongseo.ac.kr

Abstract

Unsupervised neural networks have not caught enough attention until Generative Adversarial Network (GAN) was proposed. By using both the generator and discriminator networks, GAN can extract the main characteristic of the original dataset and produce new data with similar latent statistics. However, researchers understand fully that training GAN is not easy because of its unstable condition. The discriminator usually performs too good when helping the generator to learn statistics of the training datasets. Thus, the generated data is not compelling. Various research have focused on how to improve the stability and classification accuracy of GAN. However, few studies delve into how to improve the training efficiency and to save training time. In this paper, we propose a novel optimizer, named FAST-ADAM, which integrates the Lookahead to ADAM optimizer to train the generator of a semi-supervised generative adversarial network (SSGAN). We experiment to assess the feasibility and performance of our optimizer using Canadian Institute For Advanced Research – 10 (CIFAR-10) benchmark dataset. From the experiment results, we show that FAST-ADAM can help the generator to reach convergence faster than the original ADAM while maintaining comparable training accuracy results.

Keywords: *Semi-Supervised Generative Adversarial Network, Lookahead, ADAM, Optimizer*

1. INTRODUCTION

In recent years, Convolutional Neural Networks (CNNs) have achieved great success in the area of Supervised Learning. However, unsupervised neural networks have not caught enough attention until Goodfellow and his fellow mates first proposed GAN in 2014 [1]. By using both the generator and discriminator networks, GAN can extract the main characteristic of the original dataset and produce new data with similar latent statistics. However, researchers understand fully that training GAN is not easy because of its unstable condition. The discriminator usually performs too good when helping the generator to learn statistics of the training datasets. Thus, the generated data is not compelling.

In this paper, we study the combination of ADAM and Lookahead as a new optimization algorithm to improve the convergence speed of the generator [2,3]. We choose Semi-Supervised Generative Adversarial Networks (SSGAN) as our neural network architecture [4]. In SSGAN, the authors proposed the use of semi-supervised learning and a novel feature extracting technique to improve traditional GAN. Together with our proposed optimizer, SSGAN can perform more stable and produce higher resolution images than previous

Manuscript Received: August. 24, 2019 / Revised: August. 29, 2019 / Accepted: September. 5, 2019
Corresponding Author: dkkang@dongseo.ac.kr
Tel: +82-51-320-1724, Fax: +82-51-327-8955
Department of Computer Engineering, Dongseo University, Busan, Korea

approaches.

2. PREVIOUS RESEARCH

Previous research have proposed several variants of GAN to improve the quality and robustness of the adversarial networks, such as CGAN, SGAN, SSGAN, and others.

Instead of using a specific goal to train a model, GAN is the first architecture to use neural networks to guide themselves. GAN's most significant contribution is the ability to generate a distribution of data that corresponds to the original dataset. In this way, outputs of the generator are artificial images that resemble original images but not totally the same as real images. The adversarial training method avoids direct copying or averaging of real data. Thus, increasing the diversity of generated samples.

After authors propose GAN, Conditional Generative Adversarial Networks (CGAN) emerges subsequently [5]. Traditionally, GAN is practical in training a dataset that has a low number of classes, one or two classes. However, it becomes difficult to train GAN when the number of classes increases because the output of the generator will be the mix of different images, thus, obfuscating the image quality. CGAN tackles this issue by combining noise z or real data x with label y to be in joint representation. Though the idea in this paper is preliminary, it demonstrates an efficient way to train multi-label datasets in GAN.

In 2016, Alec Radford et al. [6] proposed a class of GAN based on deep convolutional neural networks, called Deep Convolutional Generative Adversarial Networks (DCGAN). This variant is more stable than the previous GAN in most cases. They used novel structures in the generator and discriminator such as:

- Canceling any pooling layers and applying convolution layers to help DCGAN learn meaningful filters in training.
- Applying batch normalization in both the generator and discriminator to improve the speed and stability. Thus, the discriminator can classify more effectively.
- The authors also discard all fully connected layers and use distinct activation in generator and discriminator.

Compared with other unsupervised methods, DCGAN's discriminator extracts image features more efficiently and is more suitable for the image classification task.

In the same year, Tim Salimans and his lab mates focused on two techniques to improve GAN [7]. First, they proposed a virtual batch normalization, which is the extension of batch normalization in DCGAN that overcome the disadvantages of the predecessor. Second, they applied semi-supervised learning to unsupervised learning networks, which we will describe in section 3.1. Compared with other papers introducing SSGAN, their work [7] improved the techniques for training GANs and presented a detailed introduction to applying semi-supervised methods to GANs, mainly the loss functions that correspond to vanilla GAN.

3. IMPROVED SSGAN

3.1 Semi-supervised learning in GAN

In the idea of SSGAN (Semi-Supervised Learning with Generative Adversarial Networks) [4], there are N classes of labels (denoted by l_i) for training data and N output units in the form of a vector of length $N[l_1, l_2, \dots, l_N]$. We define the class of results generated by the generator G as the l_{N+1}.

According to Tim Salimans and his group's idea [7], in order to be consistent with the original GAN framework, the probability $p(y = N + 1|x)$ should be supplied to the fact that x is fake. Now we can learn not only from the $N - typed$ flagged data, but also from the G-generated data ($N + 1 - type$) after the data being considered to be labeled with $N + 1$.

3.2 Lookahead

Lookahead [3] is not a new optimization algorithm, but an algorithm that could be applied to almost all optimization algorithms and can improve its convergence speed. In traditional optimization algorithms, the weights of networks are updated by specific formula calculation A (for example, in ADAM optimizer, we update weights by using $A(L, \theta_{t,i-1,d})$ in a batch dataset). While we apply the idea of Lookahead to ADAM optimizer, we have two kinds of weights, and the name weights updated in $A(L, \theta_{t,i-1,d})$ *fast-weight*, as well as naming update A to be the inner loop.

Lookahead updates its *slow-weights* ϕ after running k times of updating *fast-weight* θ in inner loop $\phi_t \leftarrow \phi_{t-1} + \alpha_s(\theta_{t,k} - \phi_{t-1})$. Hyperparameter α_s is the learning rate of *slow-weights* and updating process of ϕ is called outer loop. As soon as *slow-weights* ϕ being updated, *fast-weight* will also be updated $\theta_{t,0} \leftarrow \phi_{t-1}$ in next step. What we will eventually use in our training process is the *fast-weight*.

3.3 Algorithm

Algorithm 1 FAST-ADAM

Input: Parameters θ

Output: Parameters θ

Initialization: first moment vector m_0, second moment vector v_0, timestep t, fast-weight $\theta_{0,0}$, slow-weight ϕ_0, synchronization period k, learning rate of fast-weight and slow-weight α_s, α_f, exponential decay rates β_1, β_2.

$m_0, v_0, t, \theta_{0,0}, \phi_0 \leftarrow 0$

$k \leftarrow 5$

$\alpha_s, \alpha_f \leftarrow 0.5$

$\beta_1, \beta_2 \in [0, 0.1]$

$L_t(\theta_{t-1})$: Loss function which is defined as the cross-entropy between observed labels and model predictive distribution $p(y/x)$.

While θ_t not converged **do**

  Synchronize parameters $\theta_{t,0} \leftarrow \phi_{t-1}$

  for $i = 1,2,\dots,k$ **do**

    $t \leftarrow t + 1$

    $g_t \leftarrow \nabla_\theta L_t(\theta_{t-1})$

    $m_t \leftarrow \beta_1 m_{t-1} + (1 - \beta_1) * g_t$

    $v_t \leftarrow \beta_2 v_{t-1} + (1 - \beta_2) * {g_t}^2$

    $m_t \leftarrow m_t / (1 - {\beta_1}^t)$

    $v_t \leftarrow v_t / (1 - {\beta_2}^t)$

    update *fast-weight:*

    $\theta_t \leftarrow \theta_{t-1} - \alpha_f * m_t / (\sqrt{v_t} + \varepsilon)$

  end for

  Update *slow-weights:*

  $\phi_t \leftarrow \phi_{t-1} + \alpha_s(\theta_{t,k} - \phi_{t-1})$

end while

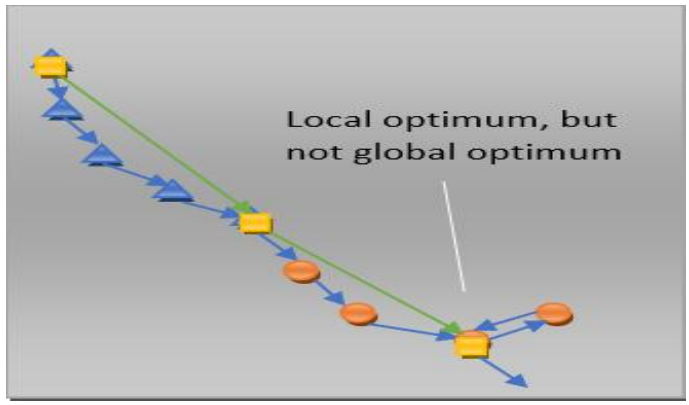

Figure 1. The blue arrows represent the update path of fast-weight, while the green arrows show the update path of slow-weight. Blue triangles are the first five updating steps, and orange balls are the second five updating steps of fast-weight. Yellow rectangles are the slow-weight's updating steps.

From Figure 1, we can see four advantages of FAST-ADAM. First, compared with ADAM optimization, FAST-ADAM usually gets a lager descent gradient resulting in fast convergence. Second, the Lookahead can help ADAM jump out of local optimum in order to explore the global optimum. Third, the effect of this approach is to reduce the variation in the training process and significantly reduce the sensitivity to suboptimal hyperparameters (thus reducing the need for a large number of attempts to adjust the parameters). Lastly, even though there are two kinds of weights needing to update, fast-weights and slow-weights will be copied or used in k steps. The computational complexity is $O\left(\frac{k+1}{k}\right)$ times of ADAM optimizer.

FAST-ADAM achieves a fast forward from the internal fast optimizer (i.e. ADAM), while there is a slower value (i.e. slow weight). While fast forward has a superior exploration effect, the slow one plays a drag-and-drop effect, that can also be seen as a stability maintenance mechanism. In general, the slow progress is in the behind, but it can efficiently drag it back (i.e. drag-and-drop) when the fast weights enter a more promisingly looking downhill but actually a local optimum. As a result, the FAST-ADAM optimizer can explore the entire space in detail without worrying about being stuck in local extrema.

4. EXPERIMENT

We ran experiments on the CIFAR10 dataset [8] to explore the improvements after we applied the FAST-ADAM. CIFAR10 dataset contains 60,000 images of 10 classes (airplane, car, bird, cat, deer, dog, frog, horse, ship, and truck). Furthermore, there are 6,000 images in each class. So, there are 50,000 training images and 10,000 test images.

We present the whole architecture of our SSGAN experiment in Table 1. We use five deconvolution layers (instead of pooling layers) in the generator. We apply Tanh as our activation function in the output layer. Meanwhile, the rest of the layers in the generator use ReLU as activation function. This idea is also the main contribution of DCGAN, in which we can learn more meaningful filters compared if we use pooling layers. The discriminator is a seven-layer deep convolutional network. The dimension of discriminator output that we use is 32*11 (from the original 32*10), because we consider the class of data generated by the generator to be class number 11.

Table 1. The architecture of the Generative Adversarial Networks

Generator			Discriminator		
Input		[32,1,1,1024]	**Input**		[32,32,32,3]
Deconv1	ReLU	[32,2,2,512]	**Conv1**	LReLU	[32,16,16,32]
Deconv2	ReLU	[32,4,4,256]	**Conv2**	LReLU	[32,8,8,64]
Deconv3	ReLU	[32,8,8,128]	**Conv3**	LReLU	[32,4,4,128]
Deconv4	ReLU	[32,16,16,64]	**Conv4**	LReLU	[32,2,2,256]
Deconv5	ReLU	[32,32,32,32]	**Conv5**	LReLU	[32,1,1,256]
Deconv6(output)	Tanh	[32,32,32,3]	**Conv6**	LReLU	[32,1,1,64]
			Conv7	LReLU	[32,1,1,11]
			Output layer		[32,11]

We mix our test data with a combination of 50% real data and 50% fake data that the generator produces. We conduct our testing every 100 epochs of training, and we present the result in Figure 2 (a).

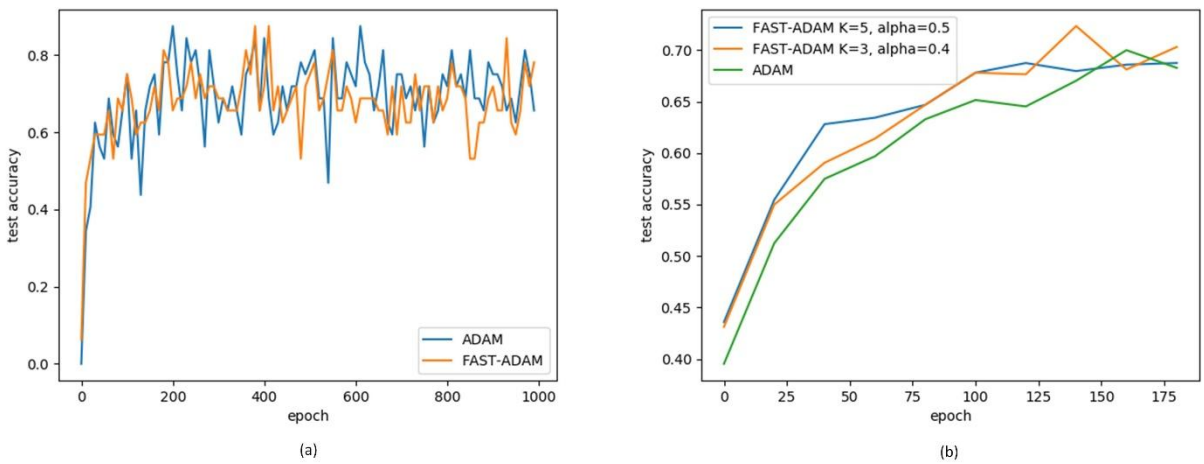

Figure 2. Experiment results. (a) test accuracy for 1000 epoch while (b) test accuracy in the first 200 epoch.

As shown in the figure, FAST-ADAM does not produce any noticeable difference in accuracy results compare to ADAM. However, we find out that the FAST-ADAM converges more quickly than the original ADAM at the first 200 epochs, as we show in Figure 2 (b). Therefore, FAST-ADAM is more efficient than ADAM as it could save training time while preserving a comparable accuracy result.

5. CONCLUSION

In this paper, we propose FAST-ADAM, a novel optimizer that combined the Lookahead and ADAM. Based on the result of our experiment using SSGAN, we observe that the Lookahead is indeed helping ADAM to converge faster in the generator network. We also tried to apply FAST-ADAM to the generator of DCGAN, but, unfortunately, we get unsatisfactory results. We conjecture that this has to do with the difference of how the discriminator performs in DCGAN and SSGAN. In DCGAN, the discriminators use labels 0 and 1 to flag whether the fed data are real images or not. However, SSGAN behaves differently such that we use N-labels that associated with the number of the class. Moreover, we also add one more label to indicate the synthesized image from the generator. In this way, the network of SSGAN can perform with a more evident goal. Thus,

when we apply FAST-ADAM to SSGAN, it can converge quickly. In our future work, we will do more research to investigate this problem and apply our algorithms to various tasks [9,10].

ACKNOWLEDGEMENT

This work was supported by Basic Science Research Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Education (NRF-2018R1D1A1A02050166).

REFERENCES

[1] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative Adversarial Nets." in Proc. Neural Information Processing Systems 2014, pp. 2672–2680, Dec. 8-13, 2014.

[2] D.P. Kingma, and J. Ba, "Adam: A method for stochastic optimization," in Proc. 3rd International Conference for Learning Representations, May 7-9, 2015.

[3] M.R. Zhang, J. Lucas, G. Hinton, and J. Ba, "Lookahead Optimizer: k steps forward, 1 step back," in Proc. Neural Information Processing Systems 2019, Poster, Dec. 8-14, 2019.

[4] A. Odena, "Semi-supervised learning with generative adversarial networks," in Proc. 33rd International Conference on Machine Learning, Workshop Paper, June 19-24, 2016.

[5] M. Mirza, and S. Osindero, "Conditional generative adversarial nets," arXiv preprint arXiv:1411.1784, 2014

[6] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," arXiv preprint arXiv:1511.06434, 2015.

[7] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, "Improved techniques for training GANs," In in Proc. Neural Information Processing Systems 2016, pp. 2234-2242, Dec. 5-10, 2016.

[8] A. Krizhevsky, V. Nair, and G. Hinton, "The CIFAR-10 dataset," online: *http://www.cs.toronto.edu/kriz/cifar.html*, 2014.

[9] G. Agrawal, and D.-K. Kang, "Wine Quality Classification with Multilayer Perceptron," International Journal of Internet, Broadcasting and Communication (IJIBC), 10(2):25-30, May 2018.
DOI: http://dx.doi.org/10.7236/IJIBC.2016.8.4.19

[10] Ho, J., and Kang, D.-K., "Ensemble-By-Session Method on Keystroke Dynamics based User Authentication," International Journal of Internet, Broadcasting and Communication (IJIBC), 8(4), November 2016.
DOI: https://doi.org/10.7236/IJIBC.2018.10.2.5