An Empirical Analysis of Generative Adversarial Network Training Times with Varying Batch Sizes

Conference Paper · October 2020

DOI: 10.1109/UEMCON51285.2020.9298092

CITATIONS
0

READS
1,378

5 authors, including:

Bhaskar Ghosh
University of Louisiana at Lafayette
12 PUBLICATIONS **47** CITATIONS
SEE PROFILE

Indira Kalyan Dutta
Arkansas Tech University
15 PUBLICATIONS **54** CITATIONS
SEE PROFILE

Albert H. Carlson
Austin Community College
27 PUBLICATIONS **43** CITATIONS
SEE PROFILE

Michael Totaro
University of Louisiana at Lafayette
46 PUBLICATIONS **320** CITATIONS
SEE PROFILE

An Empirical Analysis of Generative Adversarial Network Training Times with Varying Batch Sizes

Bhaskar Ghosh*, Indira Kalyan Dutta†
University of Louisiana at Lafayette
Louisiana, USA
*bhaskar.ghosh1@louisiana.edu,
†indira.dutta1@louisiana.edu

Albert Carlson
Kyle, Texas
ltzap1@gmail.com

Michael Totaro‡, Magdy Bayoumi§
University of Louisiana at Lafayette
Louisiana, USA
‡michael.totaro@louisiana.edu,
§magdy.bayoumi@louisiana.edu

***Abstract*—Increasing the performance of a Generative Adversarial Network (GAN) requires experimentation in choosing the suitable training hyper-parameters of learning rate and batch size. There is no consensus on learning rates or batch sizes in GANs, which makes it a "trial-and-error" process to get acceptable output. Researchers have differing views regarding the effect of batch sizes on run time. This paper investigates the impact of these training parameters of GANs with respect to actual elapsed training time. In our initial experiments, we study the effects of batch sizes, learning rates, loss function, and optimization algorithm on training using the MNIST dataset over 30,000 epochs. The simplicity of the MNIST dataset allows for a starting point in initial studies to understand if the parameter changes have any significant impact on the training times. The goal is to analyze and understand the results of varying loss functions, batch sizes, optimizer algorithms, and learning rates on GANs and address the key issue of batch size and learning rate selection.**

***Index Terms*—Generative Adversarial Networks, Training, Hyper-parameter, Neural Networks, Artificial Intelligence**

I. Introduction

Generative Adversarial Networks (GANs) were introduced by Ian Goodfellow [1] in 2014 and operate as shown in Fig. 1 [2]. GANs use a Generator Network (*G*) and a Discriminator Network (*D*) as seen in Fig. 1 to produce new samples of previously unseen data. *G* is trained to produce samples from an input noise vector and the result of the process is presented to *D*. A singular value is produced by *D* which the circuit attempts to determine whether the input data is from a real set of data or from the data produced by *G*. The output of *D* is then used as feedback to train *G* further to attempt to fool *D* into thinking that the synthetic input from *G* is instead from a real dataset.

D is commonly trained using the Stochastic Gradient Descent (SGD) [3] optimizer or the Adam optimizer [4] to minimize the loss function; however, a different optimizer might be used for alternative training scenarios. The process of training requires a great deal of data. Processing large amounts of data together can be resource and time consuming. To make this process easier, data is divided into smaller chunks known as *batches*. Batches enable the network to more easily perform calculations as the number of data points are reduced, controlling the stability of the training of a neural network as well.

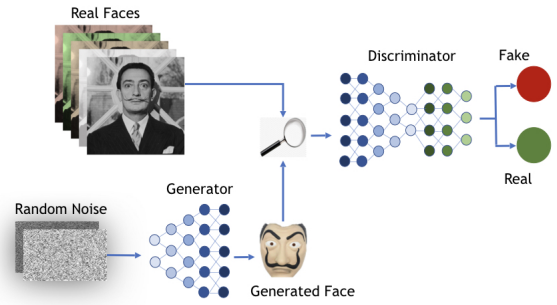

Fig. 1. Generative Adversarial Network Architecture. [2]

Batch size, or Mini Batch Size, is a static hyper-parameter that is selected by the user. However, there is no agreement on the batch size that would be most effective for the training that is to be undertaken. Small batches are assumed to converge faster and require fewer epochs [5], [6], while large batch sizes are considered better for scaling up a network [7]–[9]. To train a GAN, choosing the correct batch size is necessary to assist in training *D*, which is fundamentally a Convolutional Neural Network (CNN), and to help convergence of the GAN. Since training GANs can be time consuming, adjusting batch sizes to find the size that best suits the timing and accuracy can be challenging.

This paper attempts to address the confusion about the batch sizes. There are many hyper-parameters at play, but batch size remains the most dominant factor [10]. Batch sizes dictate the frequency of updates to the parameters within the network and controls the number of predictions made at any time. The paper explores the actual training times of a GAN with differing loss functions by training over the Mixed National Institute of Standards and Technology (MNIST) dataset [11] using a range of batch sizes. Testing procedures also explore whether or not changing the loss function plays a role in altering the training times.

II. Related Work

GANs have a *G*, trained using the output of *D*, based on a mapping of random noise from latent space, and produces

synthetic data that *D* tries to distinguish as either real or fake data. GANs use the loss function, as seen in Eq. 1, that *G* wants to minimize and *D* wants to maximize, trying to find a Nash Equilibrium [12]. The equilibrium is where the optimal outcome is achieved when no network has an incentive to change its own strategy based on the strategy of the adversary [13], while playing a Mini-Max Game. The loss function that GANs use is:

$$min_G max_D V(D,G) = \mathbb{E}_{x\sim p_{data(x)}}[logD(x)] + \mathbb{E}_{x\sim p_z(z)}[log(1-D(G(z)))] \quad (1)$$

where x is a sample from the real dataset distribution $p_{data}(x)$, z is sampled from a latent space distribution $p_z(z)$ and $D(G(z))$ is the output probability of *D* on the image generated by *G* [1]. GANs have gained success primarily in the fields of image generation and synthesis [9], [14]. GANs have also been applied by other researchers to address issues in the field of Natural Language Processing [15] and Security [16].

Neural Network (NN) training is a two step process where the goal is to minimize the cost function. The inputs are first fed into the input layer of the NN and the variables are computed in the intermediate layers, also known as *hidden layers*, completing the first step of training. This step is called the *forward pass*. The output generated from the last layer is then compared with the desired output to obtain error in gradient, which is calculated from the comparison, and, based on this comparison, weight updates are updated in each layer to minimize the error in decision making for the output. This is the second training step, known as the *backward pass*. The robustness of a network is determined by its ability to update its weights to minimize the error of the generated output. The two step training process is shown in Fig. 2 [2]. Two important terms that pertain to training a NN are *epoch* and *batches*. One *epoch* is when the NN completes training over a whole dataset. The chunks over which data is divided before training are known as *batches*.

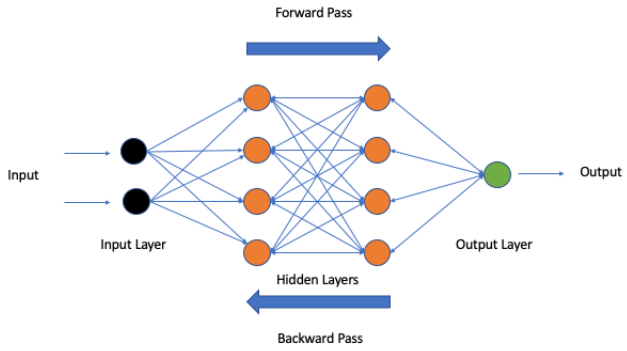

Fig. 2. Neural Network Training Process. [2]

Significant research opportunities relate to modifications to the training process. Researchers continue to search for ways to either increase convergence by changing the loss functions [17]–[19] or add Auto Encoders to increase stability of GANs. Studies have been done on batches using Normalization [10], [20], and Batch Equalization [21] for convergence of the training process; however, very few tests have been applied to the batch size selection for GAN training, thereby making it difficult to draw conclusions about the optimal selection of batch size. Batch size studies have been conducted on CNN's [22]; however, the impact of batch sizes on GANs have not been reviewed extensively. It has been shown that the effects of gradient noise on the network will impact the training times as well [6].

III. Research Goals

The primary aim of this paper is to understand the effect that batch sizes have on training the *D* with respect to elapsed training time. Along with batch sizes, the learning rate is also varied to measure the interaction effect that this will have with batch sizes to the changes in training time. We also seek to understand the interactions from varying the loss function and optimizer algorithm of the GAN will have with the change in batch sizes. In the initial experiments, with $30,000$ epochs on the MNIST dataset and a range of batch sizes, we investigate trends that researchers expect and what affects changing learning rates, optimizer algorithms and loss functions have on GAN training times.

IV. Dataset

The initial study primarily employs the MNIST dataset [11], which is a commonly used and trusted dataset in NN research [23]. The database has a collection of $60,000$ training examples and $10,000$ testing examples of 10 patterns of handwritten digit images ranging from "0" to "9" along with the labels for each image. The $10,000$ test images in the dataset are used to test the accuracy of the model. Each gray-scale image has a pixel size of 28×28 pixels containing the handwritten digit as shown in Fig.3 [11].

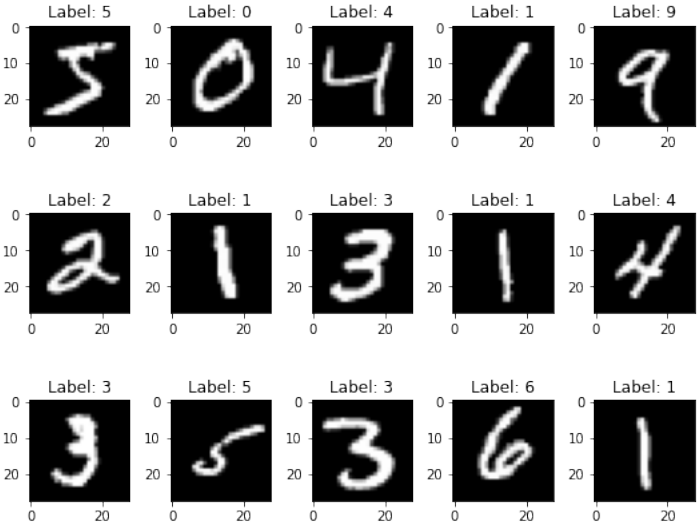

Fig. 3. MNIST Dataset with 60,000 training images and 10,000 testing images.

TABLE I
GAN TIMINGS IN SECONDS WITH DIFFERENT BATCH SIZES

	8	16	32	50	64	100	128	150	200	250	256	512	1024
GAN LR=0.01 with SGD	504.52	496.04	502.71	552.84	560.57	654.36	669.22	722.34	832.77	913.66	921.72	1401.99	2446.2
GAN LR=0.001 with SGD	478.69	475.98	483.77	530.97	537.35	615.93	643.30	698.74	806.31	895.70	886.50	1335.35	2271.98
GAN LR=0.0002 with Adam	610.76	597.13	604.81	667.62	677.27	774.01	779.54	855.20	963.34	1045.05	1054.05	1559.56	2662.36
GAN LR=0.00002 with Adam	619.16	597.95	600.39	641.57	660.10	754.21	765.70	836.78	941.02	1010.38	1015.78	1457.11	2520.13
wGAN LR=0.002 with RMSProp	859.40	842.64	876.54	941.37	971.18	1129.28	1198.75	1283.95	1488.27	1600.02	1604.52	2373.59	3919.33
wGAN LR=0.00005 with RMSProp	664.31	664.53	698.10	769.19	806.71	977.02	1047.65	1128.48	1296.38	1423.58	1440.14	2222.85	3740.96

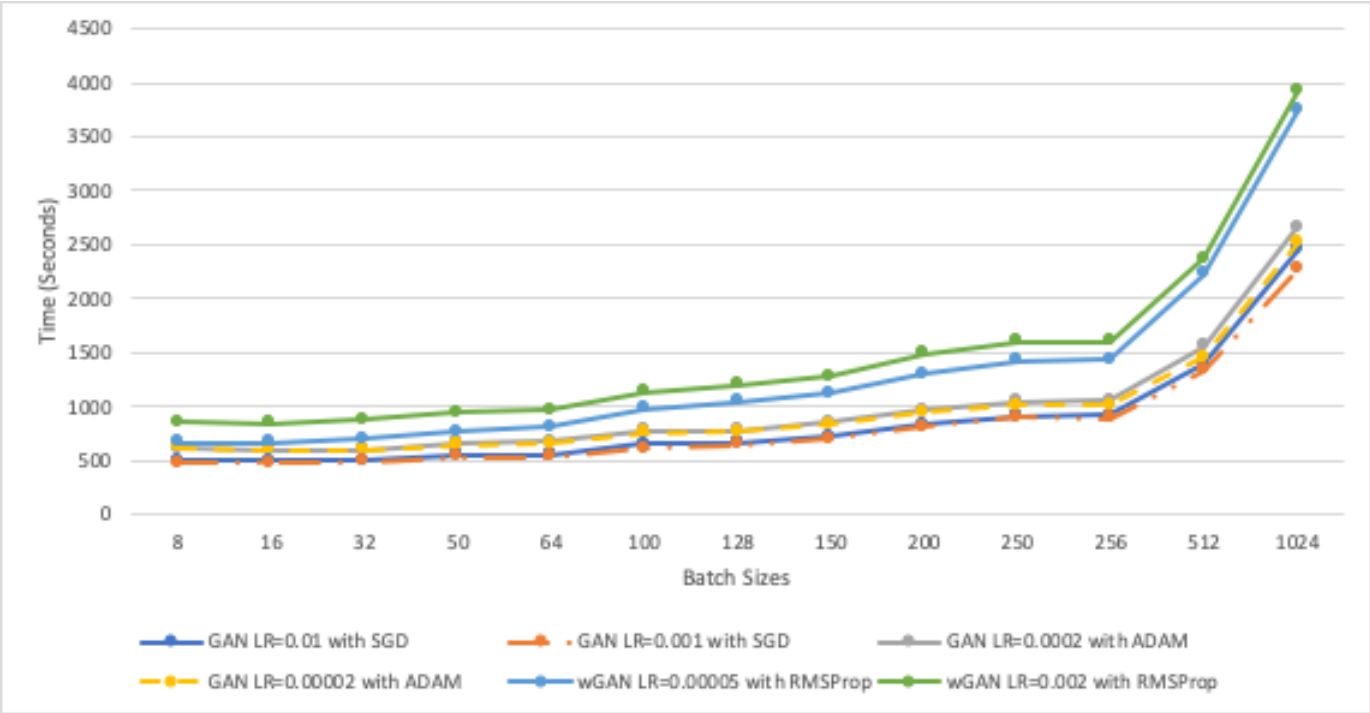

Fig. 4. Comparative training times by batch sizes, learning rates and loss functions.

V. ARCHITECTURE

The batches chosen in our experiments have sizes that are powers of two (2^n) and multiples of ten ($10n$) based on the work of Radiuk [22], with a total of 13 batches. The experiments were run on two architectures of GANs with different loss functions and optimization algorithms to understand the effects of the loss function on training time using $30,000$ epochs on a GeForce GTX 1080 Ti GPU. Keras [24] was used to complete the training, extending the code provided by [25].

A. GANs

As stated in Section II, GANs consist of a *G* and a *D*, where the aim of the *G* is to maximize the probability of the *D* incorrectly identifying the real image from a fake image. The *G* samples noise from the latent space and is converted to an image using the feedback from the *D*. *D* is a CNN, which is trained to classify fake or real images between a set of images made by *G* and a real dataset of images. In the experiment, a two layer *D* is used on varying batch sizes, first with the SGD optimizer and then with the Adam optimizer using different loss functions and learning rates.

B. Wasserstein GAN

The Wasserstein GAN [17], or wGAN, was developed by M. Arjovsky in 2017. wGAN uses a loss function called the Wasserstein-1 or the Earth-Mover (EM) distance to find the optimum path between two probability distribution functions [2], [17], which are shown in Eq. 2:

$$W(p_r, p_g) = \inf_{\gamma \in \Pi(p_r, p_g)} \mathbb{E}_{(x,y)\sim\gamma}[||x - y||] \tag{2}$$

where $\Pi(p_r, p_g)$ is the set of joint distributions $\gamma(x, y)$ with marginals p_r and p_g. $\gamma(x, y)$ is the amount of "mass" to be transported from x to y to change the distribution p_r to p_g.

The Wasserstein loss function has exhibited good stability in areas dealing well with lower dimension data distribution where Jenson-Shanon divergence, used in the vanilla GAN, has failed. The *D* in this GAN methodology turns into a critic, which scores an image with a probability value of whether it thinks the image is *real* or *fake*. The wGAN also uses a different optimizer called the Root Mean Square Propagation (RMSProp) optimizer, explained below in Section V-D.

C. Stochastic Gradient Descent

Gradient Descent [26], is an optimization algorithm that has been used successfully in Deep Learning applications. It uses an iterative method to measure the degree of change of a variable and outputs the lowest possible value of a convex cost function; however, it uses the whole set of data as one batch before updating weights of the NN. At the other extreme, SGD, the workings of which are shown in Fig. 5, addresses this issue by updating weights every iteration with a batch size of one [27].

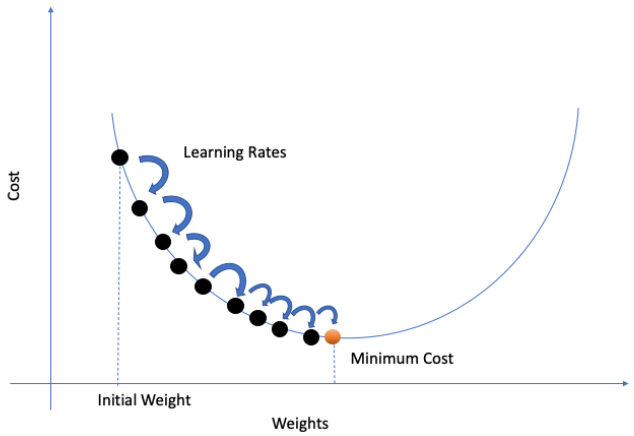

Fig. 5. Stochastic Gradient Descent.

D. RMSProp

The optimizer used by the wGAN is known as the *RMSProp* (that is, *Root Mean Square Propagation*), which is an extension of the *rprop* algorithm,developed by Geoffery Hinton [28]. The *rprop* algorithm uses a full batch for optimization to adaptively decay or speed the learning rate based on the sign of the gradient descent. This means that if a sample has the same sign on the two previous gradient updates, it speeds up the learning, since the same signs show that the learning is taking place in the right direction. Otherwise, it causes a decay in the learning rate because different signs in successive values would indicate that a step too large has taken place in the wrong direction. An issue with the rporp algorithm is that it is very time consuming when used on larger datasets. The RMSprop solves this issue by maintaining the moving average of the squared gradient of the current weights and dividing it by the square root of the mean square. This makes it possible to make learning rate adjustments on smaller batches of data [3], [29].

E. Adam

The Adam optimizer is a gradient-based optimizer that can be used interchangeably with SGD for updating the network weights iteratively [30]. The name *Adam* is imitative of the term *adaptive moment estimation*. Inspired by the *Adaptive Gradient Algorithm* (*ADAGRAD*), which maintains a learning rate for each parameter, and the RMSProp. As discussed in Section V-D, the Adam optimizer uses a flexible estimation of the "lower order moments" by calculating the exponential moving average of the gradient with the squared gradient and separate parameters, which control the decay rates [3], [30]. Note, however, that Adam is very different from the SGD, as the SGD maintains the same learning rate for each weight update [3].

F. Learning Rate

The *Learning Rate*, or *LR*, is a hyper-parameter that controls the change of the model, which is in response to weight updates based on the error of the output. It defines the rate of adaptation of a model to a problem. LR is very important as it can determine whether weight updates become "stuck" due to small rates of change, or become too fast and unstable due to high LRs. Specifically, it is the speed at which the SGD oscillates, in order to determine the lowest possible cost value, as shown in Fig. 5. The rate is often a very small positive value in the range of 0.0 and 1.0 [3].

VI. Experimental Results and Analysis

The initial experiments trained GANs using the SGD optimizer with learning rates of 0.01 and 0.001. Next, the Adam optimizer was run with a learning rate of 0.0002 and 0.00002. The loss function was then changed to characterize its effects on training by using wGAN with the RMSProp optimizer method and learning rates of 0.00005 and 0.002. All experiments were done on the MNIST dataset [11] with $30,000$ epochs. The results are shown in Table I.

Fig. 4 summarizes the initial results of the total time taken in testing the GANs. Based on the curves and the raw batch testing information, a dip was observed in run times, ranging from batch size 16 to 32. It is interesting to note that all other time values are in agreement and directly proportional to the batch size. The dip was present in every run of the test data and was consistent for that range of batch sizes. This motivates the question as to what might contribute to this sudden, short dip in training times for these values? The remainder of the curve is monotonically rising in all cases. Such a dip might be the result of two or more operations or effects that trend in different directions during training. One dominates the timing early in the training, but falls off as batch size increases, while the other has little effect early in training, but quickly dominates the training as batch size increases. If this is correct, what are the relevant factors, and what roles do they play in training?

A further question is, could this mean that there is a batch size selection that is optimal for GAN training? This question is central to future work in predicting optimal training batch sizes. The trend of timing over different GAN variants with different batch sizes are similar where training time increases with increased batch sizes.

As shown in Table I, decreasing the LR of the GAN with the SGD optimizer from 0.01 to 0.001 allows the network to converge comparatively faster. The trend of time with the decreased LR follows a similar pattern to the increased LR

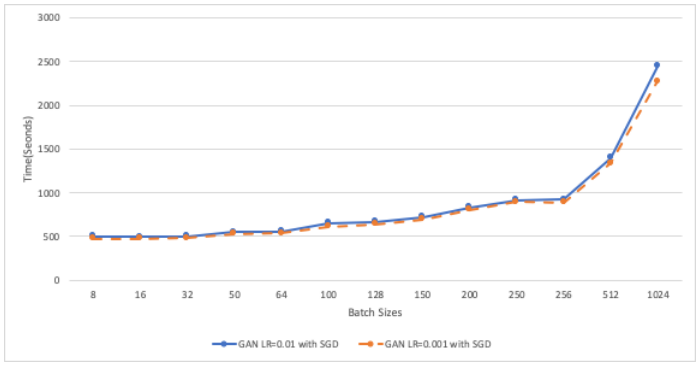

Fig. 6. Time vs Batch Sizes on GAN using SGD optimizer.

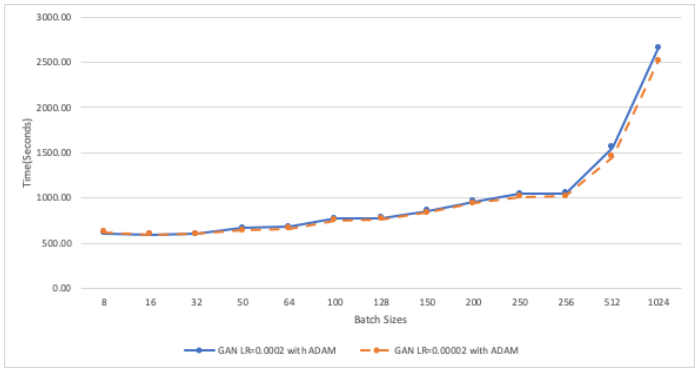

Fig. 7. Time vs Batch Sizes on GAN using Adam optimizer.

with increasing batch sizes, as shown in Fig. 6. We observe a dip in training time at batch size 256, which leads to the question as to whether a combination of batch size and/or learning rate selection has an impact to the training time (in this case). With the Adam optimizer, we observed that the starting time is high for both of the initial LRs at the first batch size. We subsequently observed a dip at batch size 16 for both LRs, which suggests that the Adam optimizer tries to update the learning rate in order to make the process more efficient. It is interesting to note, however, that for batch size 16, the time to converge is within a few milliseconds of the previous higher learning rate. This would suggest that the batch size selection plays a significant role. Convergence time increases, as shown in Fig. 7, along with the observed trend in previous experiments. For wGAN, which uses a Wasserstein loss function, the time taken to converge with a smaller LR is faster. We see in Table I that the time taken for batch size 16 is less than that for batch size 8 for the first instance, while the time is very close for the next; this follows the general trend. Thus the conclusion that batch size selection, with varying loss function, along with the smaller learning rates, helps the process to converge by reaching the lowest possible value of the cost function much faster within $30,000$ epochs. The batch size selection, along with these parameters, play a very important role as well. It is also interesting to note that the highest change in time function is between the two LRs in the wGAN; however, this had the highest change in LR.

Future work should focus on the dip in training times with respect to LRs. The training time is expected to follow the same trend of monotonic increase with batch size; however, it does not. Observe from Fig. 6 that the time taken for the 16 batch size decreases for two different learning rates. The times at batch size 256 also decreases slightly, which also raises an important question about how convergence is reached more quickly with a slower learning rate.

In Fig. 7, a batch size of 16 with the Adam optimizer tends towards a faster convergence; however, the same trend is not present in the 256 batch size. The next test shown in Fig. 8, with a very slow LR, the wGAN with RMSProp has a very small increase in time, on the order of milliseconds, with 16

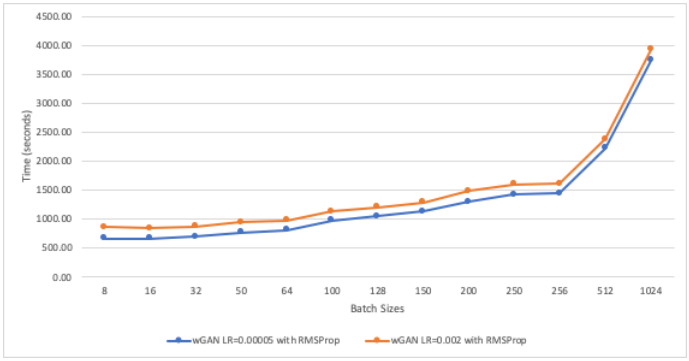

Fig. 8. Time vs Batch Sizes on wGAN using RMSProp optimizer.

batch sizes. This decrease in training time is similar to the subsequent training results, once the LR is increased to 0.002.

The aforementioned results suggest that batch size, learning rate, optimizer algorithm, and loss functions have an impact on training time. Training times are not exponential with the changes in batch size. Therefore, we cannot make a conclusive statement about the supposition that an increase in batch sizes would increase training time. Further, there are reductions in the training time at batch sizes of 16 - 32. Another small reduction in the some training time curve comes at about a batch size of 256.

VII. Conclusion and Future Research

Batch sizes and other hyper-parameters have a measurable impact on training a GAN. The optimal desired batch size is still a source of deliberation, with researchers on both ends of the batch size debate arguing their points. In this study, we investigated the effect of batch size, learning rate, optimization algorithm and loss functions on the MNIST data set. We performed our initial experiments on three different GANs. The initial tests were done over $30,000$ epochs. The results indicate that training times decrease almost consistently over batch size 16 from batch size 8 with changing LRs, optimizers and loss functions, for which training times continue to increase gradually. We also observed sudden decreases in training times, which suggests that there is an optimal batch size that can effect the "best" results from the GAN. Our

results are indicative that the optimal batch size can impact training time and convergence. The initial study results have shown good potential for an adaptive batch size that can be used as a solution to minimize training time.

This research demonstrates the impact of batch sizes and learning rate on training times. The initial assumption, as with other previous studies, was that the training time increases with increases in batch sizes; however, there is a decrease in training time with an increase in several batch sizes. In future work, we shall investigate further why there is a dip in training time, as opposed to popular and widely held assumptions about training times. It is important to understand the reasons as to why such dips occur, and then seek to predict the causes of the reduction in training time. Additionally, we intend to incorporate testing accuracy, in order to determine whether there are any changes in accuracy as well. Using another loss function might give differing results, with a different dataset. The effect of hardware could also play a role in the training time. Extending the testing to multi-GPU systems may show how training time changes with the increase in GPU size.

REFERENCES

[1] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative Adversarial Networks," *NIPS'14: Proceedings of the 27th International Conference on Neural Information Processing Systems*, vol. Volume 2, no. December 2014, pp. 2672–2680, 2014. [Online]. Available: http://arxiv.org/abs/1406.2661

[2] B. Ghosh, I. K. Dutta, M. Totaro, and M. Bayoumi, "A Survey on the Progression and Performance of Generative Adversarial Networks," in *2020 11th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*. Kharagpur: IEEE, July 2020, pp. 1–8.

[3] J. Brownlee, "Better Deep Learning. Train Faster, Reduce Overfitting, and Make Better Predictions," *Machine Learning Mastery With Python*, 2018.

[4] O. Konur, "Adam Optimizer," *Energy Education Science and Technology Part B: Social and Educational Studies*, 2013.

[5] D. Das, S. Avancha, D. Mudigere, K. Vaidynathan, S. Sridharan, D. Kalamkar, B. Kaul, and P. Dubey, "Distributed Deep Learning Using Synchronous Stochastic Gradient Descent," *ArXiv*, 2016. [Online]. Available: https://aws.amazon.com/ http://arxiv.org/abs/1602.06709

[6] S. Jenni and P. Favaro, "On stabilizing generative adversarial training with noise," *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 2019-June, pp. 12 137–12 145, 2019.

[7] P. Goyal, P. Dollár, R. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, and K. He, "Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour," *arXiv*, 2017. [Online]. Available: http://arxiv.org/abs/1706.02677

[8] Y. You, I. Gitman, and B. Ginsburg, "Scaling SGD Batch Size to 32K for ImageNet Training," *arXiv*, pp. 1–8, 2017. [Online]. Available: http://arxiv.org/abs/1708.03888

[9] A. Brock, J. Donahue, and K. Simonyan, "Large Scale GAN Training for High Fidelity Natural Image Synthesis," *36th International Conference on Machine Learning, ICML 2019*, vol. 2019-June, pp. 7392–7414, sep 2018. [Online]. Available: http://arxiv.org/abs/1903.02271 http://arxiv.org/abs/1809.11096

[10] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *32nd International Conference on Machine Learning, ICML 2015*, 2015.

[11] Y. LeCun, C. Cortes, and C. Burges, "THE MNIST DATABASE of handwritten digits," pp. 1–10, 1998. [Online]. Available: http://yann.lecun.com/exdb/mnist/

[12] J. F. Nash, "Equilibrium points in n-person games," *Proceedings of the National Academy of Sciences*, 1950.

[13] J. Eichberger, "Nash equilibrium," in *Famous Figures and Diagrams in Economics*, 2010.

[14] J. Y. Zhu, T. Park, P. Isola, and A. A. Efros, "Unpaired Image-to-Image Translation Using Cycle-Consistent Adversarial Networks," *Proceedings of the IEEE International Conference on Computer Vision*, vol. 2017-Octob, pp. 2242–2251, 2017.

[15] M. A. Haidar and M. Rezagholizadeh, "TextKD-GAN: Text Generation using KnowledgeDistillation and Generative Adversarial Networks," in *32nd Canadian Conference on Artificial Intelligence 2019*, vol. 11489 LNAI. Springer Verlag, apr 2019, pp. 107–118. [Online]. Available: http://arxiv.org/abs/1905.01976

[16] Z. Lin, Y. Shi, and Z. Xue, "IDSGAN: Generative Adversarial Networks for Attack Generation against Intrusion Detection," *ArXiv*, 2018.

[17] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein generative adversarial networks," in *34th International Conference on Machine Learning, ICML 2017*, 2017. [Online]. Available: http://arxiv.org/abs/1701.07875

[18] M. G. Bellemare, I. Danihelka, W. Dabney, S. Mohamed, B. Lakshminarayanan, S. Hoyer, and R. Munos, "The Cramer Distance as a Solution to Biased Wasserstein Gradients," *arXiv*, pp. 1–20, 2017. [Online]. Available: http://arxiv.org/abs/1705.10743

[19] X. Mao, Q. Li, H. Xie, R. Y. Lau, Z. Wang, and S. P. Smolley, "Least Squares Generative Adversarial Networks," *Proceedings of the IEEE International Conference on Computer Vision*, vol. 2017-Octob, pp. 2813–2821, 2017.

[20] S. Santurkar, D. Tsipras, A. Ilyas, and A. Madry, "How does batch normalization help optimization?" in *Advances in Neural Information Processing Systems*, 2018.

[21] W. W. Qian, C. Xia, S. Venugopalan, A. Narayanaswamy, J. Peng, and D. M. Ando, "Batch Equalization with a Generative Adversarial Network," *bioRxiv*, p. 2020.02.07.939215, 2020.

[22] P. M. Radiuk, "Impact of Training Set Batch Size on the Performance of Convolutional Neural Networks for Diverse Datasets," *Information Technology and Management Science*, vol. 20, no. 1, pp. 20–24, 2018.

[23] V. Romanuke, "Training Data Expansion and Boosting of Convolutional Neural Networks for Reducing the MNIST Dataset Error Rate," *Research Bulletin of the National Technical University of Ukraine "Kyiv Polytechnic Institute"*, 2016.

[24] Chollet François, "Keras: The Python Deep Learning library," *Keras.Io*, 2015.

[25] E. Linder-Norén, "Keras implementations of Generative Adversarial Networks." [Online]. Available: https://github.com/eriklindernoren/Keras-GAN

[26] S. Ruder, "An overview of gradient descent optimization," *ArXiv*, pp. 1–14, 2016.

[27] "ML — Stochastic Gradient Descent (SGD) - GeeksforGeeks." [Online]. Available: https://www.geeksforgeeks.org/ml-stochastic-gradient-descent-sgd/

[28] G. Hinton, N. Srivastava, and K. Swersky, "Neural Networks for Machine Learning Lecture 6a Overview of mini--batch gradient descent," University of Toronto Computer Science, Tech. Rep., 2012.

[29] "Understanding RMSprop — faster neural network learning — by Vitaly Bushaev — Towards Data Science." [Online]. Available: https://towardsdatascience.com/understanding-rmsprop-faster-neural-network-learning-62e116fcf29a

[30] D. P. Kingma and J. L. Ba, "Adam: A method for stochastic optimization," in *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, 2015.