Tornado Storm Data Synthesization using Deep Convolutional Generative Adversarial Network: Related Works and Implementation Details

Technical Report HPCF–2020–19, hpcf.umbc.edu > Publications

Carlos A. Barajas[1], Matthias K. Gobbert[1], and Jianwu Wang[2]

[1] Dept. of Mathematics and Statistics, University of Maryland, Baltimore County, Baltimore, MD 21250, USA, {barajasc,gobbert}@umbc.edu,
[2] Dept. of Information Systems, University of Maryland, Baltimore County, Baltimore, MD 21250, USA, jianwu@umbc.edu

Abstract. Predicting violent storms and dangerous weather conditions with current models can take a long time due to the immense complexity associated with weather simulation. Machine learning has the potential to classify tornadic weather patterns much more rapidly, thus allowing for more timely alerts to the public. A challenge in applying machine learning in tornado prediction is the imbalance between tornadic data and non-tornadic data. To have more balanced data, we created in this work a new data synthesization system to augment tornado storm data by implementing a deep convolutional generative adversarial network (DCGAN) and qualitatively compare its output to natural data.

Keywords: deep learning, data augmentation, data synthesization, DCGAN, TensorFlow, Keras

1 Introduction

Forecasting storm conditions using traditional, physics based weather models can pose difficulties in simulating particularly complicated phenomena. These models can be inaccurate due to necessary simplifications in physics or the presence of some uncertainty. These physically based models can also be computationally demanding and time consuming. In the cases where the use of accurate physics may be too slow or incomplete using machine learning to categorize atmospheric conditions can be beneficial [1]. Machine learning has been used to accurately forecast rain type [2, 1], clouds [2], hail [3], and to perform quality control to remove non-meteorological echos from radar signatures [4]. For more related works, see Appendix A below.

A forecaster must use care when using binary classifications of severe weather such as those which are provided in this paper. The case of a false alarm warning can be harmful to public perception of severe weather threats and has unnecessary costs. On the one hand, an increased false alarm rate will reduce the public's

trust in the warning system [5]. On the other hand, a lack of warning in a severe weather situation can cause severe injury or death to members of the public. Minimizing both false alarms and missed alarms are key in weather forecasting and public warning systems.

With advances in deep learning technologies, it is possible to accurately and quickly determine whether or not application data is of a possibly severe weather condition like a tornado. Specifically one can use a supervised neural network such as a convolutional neural network (CNN) for these binary classification scenarios. These CNNs require large amounts, hundreds of thousands and even millions, of data samples to learn from. Without an ample amount of data to learn from a CNN has no hope of achieving accurate predictions on anything except the original training data provided. Of the 183,723 storms in the data set used in this work only around 9,000 entries have conditions which lead to tornadic behavior in the future [6]. This imbalance of tornado versus no tornado results in a situation where a machine is very good at predicting no potential tornado but is very bad at predicting when there is a tornado imminent hence false negatives.

It is for these reasons there is a real motivation to acquire more data that would result in tornadic conditions. This heralds the need of synthetic data to bolster the amount of data used for training a neural network. Synthetic data must be generated such that it is indistinguishable from real data and can be used in conjunction with the natural data to train a neural network on a more balanced data set which produces less if any false negatives. Contributions of this note include: (1) We use DCGAN to synthesize tornado data to address its class imbalance challenge; (2) We conduct experiments to evaluate the feasibility of our approach qualitatively; (3) We discuss how to configure DCGAN to generate the best synthetic tornado data.

2 GAN based Data Augmentation

Each generative adversarial network (GAN) has not just one neural network but rather two networks which compete against each other to generate the best synthetic data possible. There are two parts of a GAN that make it an effective producer of synthetic data. The overall structure can be seen in Figure 1. The generator takes in random data and uses this to generate fake data. The discriminator understands what real data looks like and because of this it is capable of determining whether or not any given data is fake or real. Together these two pieces make a GAN capable of producing synthetic data similar to given data that is indistinguishable from the original data qualitatively. The process of training the GAN means providing the discriminator enough data that it can judge whether or not provided data is fake or real. Given this feedback the generator must adapt by generating more realistic data such that discriminator cannot tell the difference between the falsified data and the real natural data. If the generator can fool the discriminator, which is designed to be an expert on the data, then the synthetic data is considered just as good as naturally collected

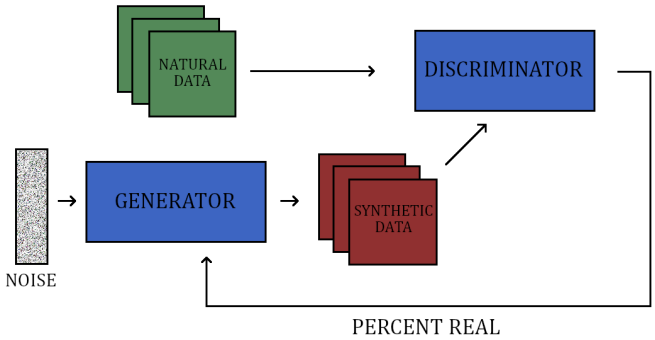


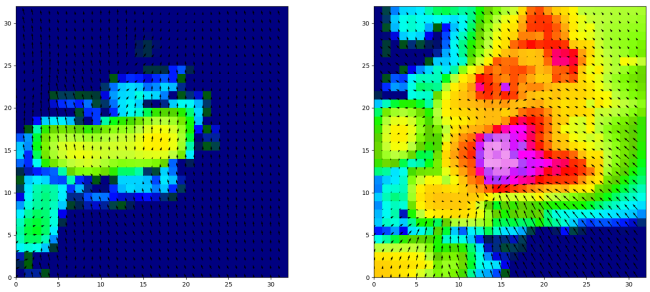
Fig. 1. The GAN is comprised of two models, the discriminator and the generator. The discriminator is training to detect whether input is fake or real and feed this output to the generator. The generator is designed to produce realistic looking data from noise based such that it passes the validation check of the discriminator.

data. Data obtained from this properly tuned GAN is typically considered more robust and a much more effective method for training than the primitive method of duplication [7].

This allows for the generation of a plethora of new data which is distinct from all previously generated data and also unique in that it is not an augmented version of any of the original data. The more interesting prospect is that, given data which is tornadic in nature, a GAN can be used to generate synthetic data that is also promised to be tornadic [8]. With a GAN, new original data can be generated for the training of the CNN that would be used for prediction. This CNN when given real natural data from an upcoming storm would be able to accurately and instantly deliver a verdict of tornadic or not, rather than waiting for a simulation to finish days later.

3 Data

The data set used in this analysis was obtained from the Machine Learning in Python for Environmental Science Problems AMS Short Course, provided by David John Gagne from the National Center for Atmospheric Research [9]. Each file contains the reflectivity, 10 meter U and V components of the wind field, 2 meter temperature, and the maximum relative vorticity for a storm patch, as well as several other variables. These files are in the form of $32 \times 32 \times 3$ images describing the storm. We treat the underlying data as an image and push it through the CNN as if it were a normal RGB image. This allows our findings to generalize to other non-specialized CNNs. Figure 2 shows two examples image from one of these files. Storms are defined as having simulated radar reflectivity of 40 dBZ or greater as seen in Figure 2 (b). Reflectivity, in combination with the wind field, can be used to estimate the probability of specific low-level vorticity speeds. In the case of Figure 2 (a) the reflectivity and wind field were not



(a) (b)

Fig. 2. Sample images of radar reflectivity and wind field for a storm which (a) does not and (b) does produce future tornadic conditions.

sufficient enough to cause future low-level vorticity speeds. The dataset contains nearly 80,000 convective storm centroids across the central United States.

We preprocessed the original NCAR storm data containing 183,723 distinct storms, each of which consists of $32 \times 32 \times 3$ grid points, and extracted composite reflectivity, 10 m west-east wind component in meters per second, and 10 m south-north wind component in meters per second at each grid point giving approximately 2 GB worth of data. We use the future vertical velocity as the output of the network. This gives us 3 layers of data per storm entry producing a total data size of $183{,}723 \times 32 \times 32 \times 3$ floats to feed into the neural network. We use 138,963 storms for training the model and 44,760 storms for testing the accuracy of the model. We track the total wall time for training and testing over both image sets.

With only a handful of tornadic cases present in the base dataset we used primitive augmentation techniques to bolster the number of tornadic cases to be fed into the DCGAN. The only primitive augmentation technique used was rotation. Reflection and translation could generated a storm that might not physically possible.

4 Results: DCGAN based Weather Data Synthesization

The numerical studies in this work use one GPU node from 2018 containing four NVIDIA Tesla V100 GPUs connected by NVLink. The node has 384 GB of memory (12×32 GB DDR4 at 2666 MT/s) connected through two 18-core Intel Xeon Gold 6140 Skylake CPUs (2.3 GHz clock speed, 24.75 MB L3 cache, 6 memory channels).

The input data was the original input data inflated by primitive augmentation techniques covered in Section 3. The constants for training were: learning rate of 0.001, batch size of 128, data multiplier of 1, and 1 GPU. The images produced by the generator are logged every 25 epochs and evaluated qualitatively

as they improve in realism. For more implementation details, see Appendix B below.

First for reference recall that Figure 2 contains examples of a tornadic and non-tornadic image. All of the images in example images should be the base of which these qualitative comparisons happen. A layperson would observe the tornadic case and say that there are clearly centers of activity with regards to reflectivity and wind patterns relative to that activity. However note that this is just one storm that happened to look like this and that many of the other tornadic cases have several areas of high reflectivity. Take note that for both the tornadic and non tornadic images transitions from high reflectivity to low reflectivity are smooth with clear edges of transition.

Consider Figure 3 which contains several rows of images. Each row represents three images generated at the listed epoch number. The first row of the figure contains three images generated before any training was done. whereas the last row represents three images generated after all the training had been done. The images in the first row are more like noise than real weather which is to be expected as the generator takes in raw noise from a Gaussian distribution. The second row of images are from the 25 epoch marker. Each image has some of the hallmarks of tornadic storm. There are clear attempts at nesting reflectivity levels such putting higher reflectivity in centers or groups. Yet the generator has not been able to really gauge how sensitive the ranges should be and is mixing high and low reflectivity where one might expect the centers to be areas with the most reflectivity. Additionally the wind velocities appear to be very random and non-sensical. At 50 epochs the generator has learned how to more properly gauge relative reflectivity levels. The sizes of the high reflectivity clusters seem very small and uneventful. Additionally there is not enough variation in reflectivity. The images are just mostly high levels of reflectivity rather than concentrated area of high reflectivity which transitions to very low reflectivity over time. The wind patterns are being created in ways that they are moving in ways relative to the clusters of reflectivity which is a positive sign.

75 and 100 epochs are where the generator has a solid grasp of what it should be doing and the differences between these stages are subtler than previous ones. The storms for 75 and 100 epochs have a rich variation of high and low reflectivity especially when compared to previous epoch counts. Each has a clear set of centers that smoothly transition into lower reflectivity as you move away them. The wind patterns have a clear dependence on the centers position and intensity. This is especially apparent when compared to previous epoch markers. At 75 epochs the storms were either very mundane or had a single high activity center or two. The majority of the storms looked well formed but still not as distinct as would be expected. The 100 epoch benchmark produced the most varied and interesting data set. All of the storms have different shapes, sizes, intensities, and centers. There is a clear and smooth transition from high reflectivity to low reflectivity. The wind patterns seem to interact with the centers and have purpose.

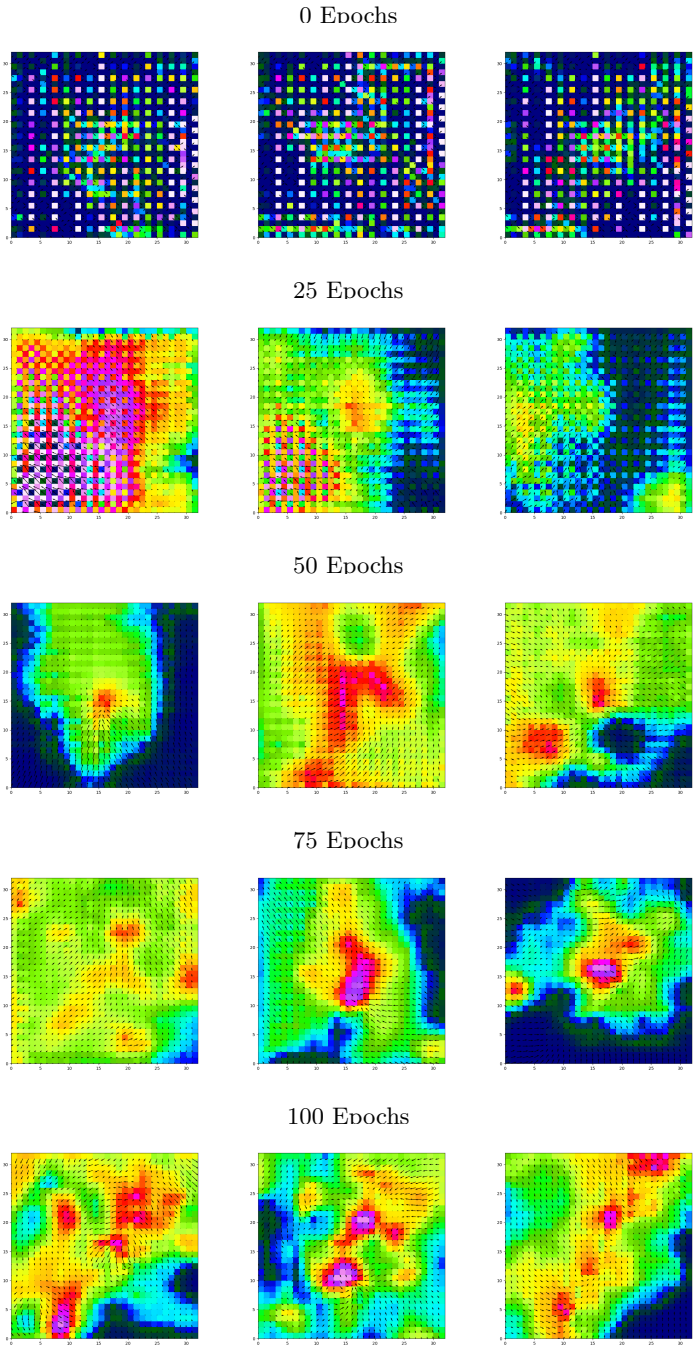


Fig. 3. Sample images of GAN generator output at 0, 25, 50, 75, 100 epochs.

5 Conclusions

The DCGAN is the first complex network trained using the new framework from [6]. The images started out as nothing more than plottable noise. As the epochs progress every new set of generated images slowly gained additional qualities that put them closer to the realm of realism. At the 75 epoch mark most of the storms were mostly indistinguishable from the real storms in the data set, however there were small features which were still not accurate to the careful eye of a layperson. At 100 epochs the images were completely indistinguishable from the real storm data set by a layperson. The transitions, the realistic concentrations of reflectivity, the obvious correlation of wind velocity relative to the concentrated activity, and the rich variety of storms produced were all quality. The data set might be able to be used to train the predictive network to predict real storms as the data is indistinguishable by the casual observer.

Acknowledgment

This work is supported in part by the U.S. National Science Foundation under the CyberTraining (OAC–1730250) and MRI (OAC–1726023) programs. The hardware used in the computational studies is part of the UMBC High Performance Computing Facility (HPCF). Co-author Carlos A. Barajas was supported as HPCF RA as well as as CyberTraining RA.

References

1. Nourani, V., Uzelaltinbulat, S., Sadikoglu, F., Behfar, N.: Artificial intelligence based ensemble modeling for multi-station prediction of precipitation. Atmosphere **10**(2), 80–27 (2019)
2. Ghada, W., Estrella, N., Menzel, A.: Machine learning approach to classify rain type based on Thies disdrometers and cloud observations. Atmosphere **10**(251), 1–18 (2019)
3. McGovern, A., Elmore, K.L., Gagne II, D.J., Haupt, S.E., Karstens, C.D., Lagerquist, R., Smith, T., Williams, J.K.: Using artificial intelligence to improve real-time decision-making for high-impact weather. Bulletin of the American Meteorological Society **98**(10), 2073–2090 (2017)
4. Lakshmanan, V., Karstens, C., Krause, J., Elmore, K., Ryzhkov, A., Berkseth, S.: Which polarimetric variables are important for weather/no-weather discrimination? Journal of Atmospheric and Oceanic Technology **32**(6), 1209–1223 (2015)
5. Barnes, L.R., Gruntfest, E.C., Hayden, M.H., Schultz, D.M., Benight, C.: False alarms and close calls: A conceptual model of warning accuracy. Weather and Forecasting **22**(5), 1140–1147 (2007)
6. Barajas, C.A., Gobbert, M.K., Wang, J.: Performance benchmarking of data augmentation and deep learning for tornado prediction. In: 2019 IEEE International Conference on Big Data (Big Data), pp. 3607–3615. IEEE (2019)
7. dos Santos Tanaka, F.H.K., Aranha, C.: Data augmentation using GANs. ArXiv **abs/1904.09135** (2019)

8. Bok, V., Langr, J.: GANs In Action. Manning (2019)
9. Lagerquist, R., Gagne II, D.J.: Basic machine learning for predicting thunderstorm rotation: Python tutorial. https://github.com/djgagne/ams-ml-python-course/blob/master/module_2/ML_Short_Course_Module_2_Basic.ipynb (2019)
10. He, H., Garcia, E.A.: Learning from imbalanced data. IEEE Transactions on Knowledge and Data Engineering **21**(9), 1263–1284 (2009)
11. C., N.V., et al.: SMOTE: Synthetic Minority Over-sampling Technique. Journal of Artificial Intelligence Research **16**, 321–357 (2002)
12. Stefanowski, J.: Dealing with data difficulty factors while learning from imbalanced data. Challenges in Computational Statistics and Data Mining **605**, 333–363 (2016)
13. Wang, S., Li, Z., W.Chao, Cao, Q.: Applying adaptive oversampling technique based on data density and cost-sensitive SVM to imbalanced learning. In: Proceedings of the International Joint Conference on Neural Networks, pp. 1–8 (2012)
14. Zhang, W., Wang, J.: A hybrid learning framework for imbalanced stream classification. In: 2017 IEEE 6th International Congress on Big Data, pp. 480–487 (2017)
15. Barajas, C.A.: *An Approach to Tuning Hyperparameters in Parallel: A Performance Study Using Climate Data*. M.S. Thesis, Department of Mathematics and Statistics, University of Maryland, Baltimore County (2019)
16. Schwartz, C.S., Romine, G.S., Weisman, M.L., Sobash, R.A., Fossell, K.R., Manning, K.W., Trier, S.B.: A real-time convection-allowing ensemble prediction system initialized by mesoscale ensemble kalman filter analyses. Weather and Forecasting **30**(5), 1158–1181 (2015)
17. Bowles, C., Chen, L., Guerrero, R., Bentley, P., Gunn, R., Hammers, A., Dickie, D.A., Hernández, M.V., Wardlaw, J., Rueckert, D.: Gan augmentation: Augmenting training data using generative adversarial networks (2018)
18. Sandfort, V., Yan, K., Pickhardt, P.J., Summers, R.M.: Data augmentation using generative adversarial networks (cyclegan) to improve generalizability in ct segmentation tasks. Scientific reports **9**(1), 1–9 (2019)
19. Chen, K., Zhou, X., Xiang, W., Zhou, Q.: Data augmentation using gan for multi-domain network-based human tracking. In: 2018 IEEE Visual Communications and Image Processing (VCIP), pp. 1–4. IEEE (2018)
20. dos Santos Tanaka, F.H.K., Aranha, C.: Data augmentation using gans (2019)

A Related Works

Class Imbalance is a well known challenge in machine learning and a large number of approaches have been proposed with respect to this specific issue. These methods fall into three main categories: data-level solutions [10] using oversampling and undersampling techniques such as SMOTE [11], algorithm-level solutions [12], and hybrid solutions [13, 14] are proposed that combine the strong points of previous two methods to address classification with uneven data representation. Our work falls in the first category and addresses Class Imbalance via data argumentation of the minority class. As our previous study at [15] shows, common oversampling and undersampling techniques only work for 1D data, not 2D data. So our work in [15] worked on data argumentation via perturbation of original images, such as rotations, zooming and flipping.

In climate/weather domain, data could be augmented via simulation models, such as [16]. These are very computationally expensive, often taking days for only a few hours of simulated data. On top of that there are variations between each of the models used to simulate storms, each with their own meaningful results and possible drawbacks. The computational cost of these models and the time to generate the synthetic data is what gives machine learning based data argumentation an edge. If a storm can be predicted without the need for simulations, because a machine learning model takes raw satellite data and quickly produces a prediction, then solving the data imbalance for the initial training gives machine learning based approach a clear advantage.

GAN based image data augmentation techniques have been used for medical images, e.g., [17, 18] and human images, e.g., [19, 20]. Yet we have not seen many studies on GAN techniques being useful for weather data argumentation. We believe our work is the first to apply DCGAN to tornados.

B Storm Synthesization with DCGAN

The DCGAN used in Section 4 is covered in this section. The exact structure of the DCGAN varies slightly from the typical GAN by using an alternative final activator for the discriminator. The layer setup for discriminator, generator, and GAN structure are detailed in Section B.1. Additionally the actual training of the GAN is handled by the parallel framework in [6] but the standard fitting methods implemented by Keras are not enough for training the DCGAN. To prevent changes to the framework a pseudo-subclass of Keras.Sequential, referred to as wGAN, is used that comes with a model building method and a custom fitting method which are detailed in Section B.2.

B.1 Structure of the DCGAN

The DCGAN is made up of two separate neural networks: the discriminator which is a CNN and the generator which is also a CNN. All layers are merged together using a Keras.Sequential object and all layer names are directly from the Keras.layers api. The discriminator is based on the proposed DCGAN discriminator in [8]. The input layer takes a $32 \times 32 \times 3$ image and pushes it through a Conv2D layer converting it into a $16 \times 16 \times 32$ tensor followed by a LeakyReLU activator with $\alpha = 0.01$. The output is then fed into a Conv2D layer converting the layer input into a $8 \times 8 \times 64$ tensor which is immediately piped into a Keras BatchNormalization layer. The BatchNormalization layer helps keep each intermediate result between the layers normalized to prevent problems with outliers from arising in between layers. The BatchNormalization layer is then connected to a LeakyReLU activator with $\alpha = 0.01$. Then the last Conv2D layer converts the layer input into $4 \times 4 \times 128$ tensor the same BatchNormalization layer and LeakyReLU as mentioned for the previous layers. The kernel size and square stride length for all Conv2D layers is 3 and 2 respectively. Then the final Conv2D

result is sent to a Flatten layer with a Sigmoid activator. The Sigmoid activator allows for better learning as it returns a what we consider to be a percent chance of the input being real rather than the traditional binary classification of real and not real [8]. The model uses the Adam optimizer and computes loss as binary crossentropy.

The generator is designed in a similar but reverse fashion and is similar to the one for a DCGAN in [8]. The input layer for the generator takes in a 100×1 vector. The vector is pushed through a fully connected Dense layer with 32,768 neurons and reshaped into into a $8 \times 8 \times 512$ tensor. This input is fed into a Conv2DTranspose layer which converts it to a $16 \times 16 \times 256$ tensor with a square stride of 2 and a kernel size of 3. Then a BatchNormalization layer followed by LeakyReLU activator as discussed in the discriminator. The following layer is another Conv2DTranspose layer which converts the layer input into a $16 \times 16 \times 128$ tensor with a square stride 1, to prevent a change in dimensions, and a kernel size of 3. This is followed by a BatchNormalization layer with a LeakyReLU. Then the final Conv2DTranspose takes the previous tensor input and converts it into a $32 \times 32 \times 3$ image with a square stride of 2 and a kernel size of 3. The final activator is hyperbolic tangent which is known to result in better image quality [8]. This also means that all images output by the generator will be compressed to numbers from -1 to 1 and will need to be reinflated for real use and evaluation.

The final GAN structure puts the generator into a Keras.Sequential object first, followed immediately by the discriminator whose trainability is disabled. The discriminator serves as a judge of the generators creative prowess so its knowledge must remain static while the generator is trained via this combined structure. The GAN model uses the Adam optimizer and computes loss as binary crossentropy.

In order to conform with the parallel training frameworks API the creation of the discriminator, generator, and GAN are all wrapped inside a python object which acts a pseudo-subclass of the Keras.Sequential object. The training process that the GAN must undergo is not the same as a traditional neural network as the generator and discriminator must be trained entirely separate while maintaining their connectedness. Thus a custom fitting object named wGAN is written such that the parallel framework does not have to change its interactions with model but we can still do the special training process required.

B.2 Training the DCGAN with wGAN

The fitting object will be referred to as wGAN for easy reference. The wGAN's fitting method takes all the same inputs as the regular Sequential.fit and is our solution to training the DCGAN in an HPC compliant way. The wGAN first isolates the data which result in tornadic conditions and attempts to honor the original batch size provided. If there are not data to honor the batch size it opts for all tornadic data to be used. The wGAN randomly rearranges the order of the images for each epoch as is customary for training neural networks to prevent training via familiarity. All images are compressed to the range $[-1, 1]$ so that

discriminator can sensibly compare the outputs of the generator to the real images. For one epoch a batch of these rearranged images is considered. These images are marked as 1 for real. The generator is then fed as many 100×1 Gaussian noise vectors as the number of real images being used. These fake images are tagged as 0 for fake. The discriminator is then trainined to recognize the real images as real. Then the discriminator is fed the fake images and told that they are all fake. The generator is then given as many 100×1 noise vectors as the number of real images being used and its output images and funneled to the discriminator. A batch of real data is also provided to the discriminator in additional to the output of the generator. The discriminator reports a percent chance that each image provided is real. The generator uses this feedback to compute weights in a manner such that next time, it will come closer to fooling the discriminator. This process is repeated for the provided number of epochs. Every 25 epochs 10 of the generated images are reinflated to their intended values and saved to disk. After all epochs have been completed 100 generates images are reinflated to their intended values and saved to disk.