USING GANS TO PRODUCE ART IN A PARTICULAR STYLE FROM SEMANTIC MAPS

MATHEW TUCKER

Abstract. We investigate the role of noise in the training process of an image-to-image style transference GAN. We will do this by comparing two models trained on the same training data, a set of semantic maps and their target image; however, in one of the models, we will add noise to the semantic maps prior to the training process. We will then apply our models to a semantic map that was not used in the training process and measure the Euclidean distance, treating the RGB pixel values as spacial coordinates, of the generated image from its target image to compare the accuracy of our two models.

Date: December 10, 2022.

1. Introduction

Consider the following scenario and question,

> a car company, with limited resources, is working on building a self-driving car. The company is struggling to get the permissions necessary to drive the car on open roads to collect the data necessary to train the AI that steers the car.

How might they get the data they need?

One method might be to collect the data using human drivers by equipping a collection of their employees' cars with an array of sensors to collect the data. But that would result in the data being biased to the road conditions of just that region. Another solution they could implement is to employ drivers in different regions around the world to collect the necessary representative data. They could then, using this data, apply generative adversarial networks(GANs) to generate the quantity of data necessary to train their AI. Methods such as the one just suggested, where GANs are trained on existing data to generate new data, have been applied in a number of fields ranging from medicine to financial data. Given that, and the growing role that other machine learning technologies are playing in society today, the importance of these technologies should be clear. It is with that in mind that we will set out to understand the basic architecture of a GAN as well as the fundamental mathematics behind them. We will also, investigate the role of noise in training data and how it might be possible to leverage noise to improve the quality of training.

The GAN[1] architecture was established in 2014 when Goodfellow et. al. published a paper showing it to be a significant improvement on other generative models when applied to the the MNIST, and other, data sets. Since then we have seen an explosion of research on GANs leading to the development of other kinds of GANs. Unfortunately, we can't do a complete survey of these other styles of GANs, so we will just mention a few here with a brief description:

- DCGAN
  - some of the deviations from a GAN that are implemented by DCGANs:
    - the discriminator uses Leaky ReLu for its action potential at ever layer
    - both use batch normalization
    - the generator use Leaky ReLu for every layer except the last one
    - implements transposed convolutional network in the generator
    - neither the generator nor the discriminator use max pooling
  - Maps random input to a target
- WGAN
  - implements "Wasserstein distance" to measure the difference between the probability distribution of the generator and the probability distribution of real images.
  - Maps random input to a target
- Conditional GAN
  - adds the requirement that something must be specified about the desired outcome. In other words, the generator of a Conditional GAN's models conditional probability $P(x|y)$ while a GAN models the joint probability $P(x, y)$.
  - Maps random input AND a condition(goal) to a target
- pix2pix
  - here the generator implements what is known as a u-net and the discriminator uses what the authors, of the paper that established pix2pix, coined as a PatchGAN.
  - Maps an image to an image

A few other styles of GANs are the CycleGAN, DiscoGAN, StyleGAN, and SRGAN. Again, clearly the GAN eco system is robust and just too extensive to do any justice too here. As a result we will spend a bulk of this paper reviewing, in broad strokes, the basic GAN as originally defined by Goodfellow, et. al. We will then go on to explain the setup of our experiment, the training process, the results of our experiment, and a brief discussion. To begin, let's quickly review the architecture employed by a GAN.

[1]It's worth mentioning, this original style of GAN is sometimes called a VanillaGAN.

2. GANs

A GAN is a machine learning architecture that implements two different deep neural networks, one is dubbed the generator and the other is the discriminator. The way these two networks work can be thought of as being analogous to a counterfeiter(the generator) and a police officer(the discriminator), where the goal of the training process is for the counterfeiter to learn the distributions associated with the real data so that they can generate forgeries that police officer can't tell apart from the real thing. What that looks like in practice, however is a bit more complicated.

As the reader may have noticed from the description above, a GAN can be framed as a game where the two players(the counterfeiter/generator and the police officer/discriminator) are both trying to minimize their opponents payoff while maximizing their own. Another way of saying this, "is that a player is minimizing their opponents maximum payoff" or "that a player is trying to minimize their loss in a worst case scenario." Formally this can be written as,

$$\overline{u_i} = \min_{s_{-i}} \max_{s_i} u_i \left(s_i, s_{-i}\right)$$

where

- i is the index of the player of interest,
- $-i$ denotes all other players except player i,
- s_i is the strategy taken by player i,
- s_{-i} denotes the strategy taken by all other players,
- and u_i is the payoff/utility/value function of player i.

This is often referred to in the literature as a minimax[2] game. As a note, this is a bit of a misnomer as a "minimax game" refers to a situation where both players are playing this same strategy(they could just as easily apply some other strategy and it no longer be a minimax game), and not to a particular kind of game like a "game against nature."

Now, notice the actual value of the payoff functions is actually rather arbitrary. This is because, in our case, we only care that the results of our game are ordered since we're not talking about "intelligent actors" with the ability to choose to play anything other than the "pure strategy"[3] of our choosing. For this reason, we can assign the players payoff functions which are most convenient for us, in particular we will assign them the payoff functions,

$$\begin{cases} U_G = \mathbb{E}_{z\sim p_z(z)} \log\left(1 - D[G(z)]\right) \\ U_D = \mathbb{E}_{x\sim p_{data}(x)} \left(\log D(x)\right) + \mathbb{E}_{z\sim p_z(z)} \log\left(1 - D[G(z)]\right) \end{cases}$$

where

- x is the real data,
- $G(z)$ is the data generated from some random value z,
- $D(\cdot)$ is the discriminator's confidence that the provided data is real,
- p_z is the generated distribution,
- and p_{data} is the distribution of the data.

But what makes these values so favorable? Why is it, we should choose this system of equations to be the payoff functions? The reason lies with U_D, the discriminators payoff function and its relationship to binary cross entropy. For more about that relationship, see slides 22-25 of the accompanying presentation.

[2]The author should note, I don't know of an instance when minimax is applied by name to a nonzero sum game within the strict context of Game Theory. That said, in the context of a Game Theory text/course, I have seen the process be applied to nonzero sum games just not referred to by name in those instances. Moreover, when I went looking for examples, I did find several applications of minimax to nonzero sum games where it was referred to by name in material related to CS. This, and the fact that I found a paper by a game theorist making the point that the game described was not zero sum, leads me to believe that there is a slight difference in how the game theorists and computer scientists use the phrase "minimax."

[3]In Game Theory, when a player decides to play only one strategy, we call this a pure strategy. In the event that player decides to play a combination of their pure strategies during a game, we call this a "mixed strategy;" and, we assign probabilities to each of the pure strategies to denote how frequently they are played.

Now, that we've got our payoff functions defined, let's talk about how a GAN uses them to "learn."To learn, GANs implement what is known as backpropagation[4] by first taking the gradient of the binary cross-entropy loss functions who's terms are defined by the the payoff functions. But how do we know that an optimal solution exist? And when, if ever, does our algorithm[5] converge?

3. Proofs and Convergence

Before we go any further, we should ask ourselves, how do we know that an "optimal" solution exist? We will prove this in two parts. In the first part we will show the optimal result for $D(x)$ is, $D^*(x) = \frac{p_{data}}{p_{data}+p_z}$. In the second part we will show that, under the condition $D(x) = D^*$, $p_z = p_{data}$ is our global minimum.

Proof:
For a fixed generator G, consider the payoff function to the discriminator

$$U_D = \mathbb{E}_{x\sim p_{data}(x)} \left(\log D(x)\right) + \mathbb{E}_{z\sim p_z(z)} \log\left(1 - D[G(z)]\right)$$

Expanding this out we have,

$$U_D = \int_{\Omega(x)} p_{data}(x) \log D(x) dx + \int_{\Omega(z)} p_z(z) \log\left(1 - D[G(z)]\right) dz,$$

where $\Omega(\cdot)$ is the domain of that variable. Since we are maximizing the system here, we assume $x = z$ and find that we can reduce the expression to the single integral

$$U_D = \int_{\Omega(x)} p_{data}(x) \log D(x) + p_z(x) \log\left(1 - D[G(x)]\right) dx.$$

But, observer $(p_{data}(x),\ p_g(x)) \in (0,\ 1] \times (0,\ 1]$ and for $(a, b) \in \mathbb{R}^2/(0,\ 0)$ we know that $a\log(y) + b\log(1-y)$ is maximal iff $y = \frac{a}{a+b}$. Hence, U_D is maximal only when $D(x) = \frac{p_{data}}{p_{data}+p_z}$. □

In keeping with the original paper, let's now observe, for $D^*(x) = \frac{p_{data}}{p_{data}+p_z}$ we can reframe U_D as

$$C(G) = \mathbb{E}_{x\sim p_{data}(x)} \left(\log \frac{p_{data}(x)}{p_{data}(x) + p_z(x)}\right) + \mathbb{E}_{z\sim p_z(z)} \log\left(1 - \frac{p_{data}(x)}{p_{data}(x) + p_z(x)}\right).$$

So U_D now depends entirely on the generator, G. Let's also recall, from the presentation, that

$$D_{KL}(p||q) = \mathbb{E}(\log \frac{p}{q}) = \sum_{\forall x\in X} p(x) \log\left[\frac{p(x)}{q(x)}\right].$$

Finally, before moving on to the proof that $C(G)$ is minimal at $p_{data} = p_z$, it should be stated that the Jensen–Shannon divergence is,

$$D_{JS}(p||q) = \frac{D_{KL}(p||\frac{p+q}{2}) + D_{KL}(q||\frac{p+q}{2})}{2}.$$

Now, we wish to prove $C(G)$ is minimal at $p_{data} = p_z$.

Proof:
Observe, by the addition of the zero, $\log 4 - \log 4$ to $C(G)$ we get that

$$C(G) = D_{KL}(p_{data}||p_{data} + p_z) + D_{KL}(p_z||p_{data} + p_z) + \log 4 - \log 4.$$

But we can break $\log 4$ into two equal terms and add them to our Kullback–Leibler divergence terms to get

$$C(G) = D_{KL}(p_{data}||\frac{p_{data} + p_z}{2}) + D_{KL}(p_z||\frac{p_{data} + p_z}{2}) - \log 4 = D_{JS}(p_z||p_{data}) - \log 4.$$

[4]Essentially this is just an application of chain-rule to update the weights of a neural net. However, for more information on backpropagation(and neural nets), I encourage the reader to start by watching the last two videos in 3Blue1Brown's video series on neural nets, https://youtube.com/playlist?list=PLZHQObOWTQDNU6R1_67000Dx_ZCJB-3pi

[5]See algorithm 1 in section 9

And by properties of the Jensen–Shannon divergence[6] it follows that $C(G)$ achieves its global minimum whenever $p_z = p_{data}$. □

In the paper by Goodfellow et.al., the authors go on to conclude the following proposition

Proposition:
If G and D have enough capacity, and at each step of Algorithm 1, the discriminator is allowed to reach its optimum given G, and p_g is updated so as to improve the criterion

$$\mathbb{E}_{x\sim p_{data}(x)}\left(\log D(x)\right) + \mathbb{E}_{z\sim p_z(z)}\log\left(1 - D[G(z)]\right)$$

then p_g converges to p_{data}. □

The proof for this will be left as an exercise for the reader, though it is covered in the original paper.

4. The Model and Noisy Data

To see why the author suspects noise may aid in the training process, it may help if we consider a simplified analogy. Let's consider the function,

$$f(x) = \frac{\sin\left(\frac{\pi}{2}x\right)}{x}, \qquad x \in [1, \infty]$$

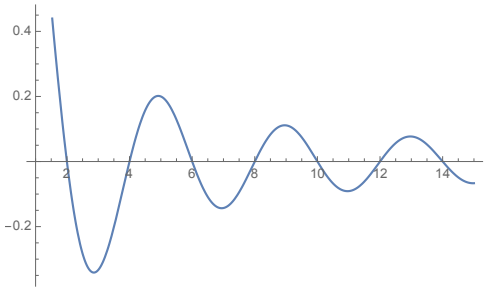

FIGURE 1. Graph of $f(x)$.

Clearly, this function has a plurality of local minimums and maximums. So if we were to devise a method of traversing our function to find minimums, how would we know which is the absolute minimum? That is to say, how can we be sure that our algorithm converged onto a proper solution? Let's assume we've implemented a method of finding minimums and it converges to $x = 6.9417\cdots$.

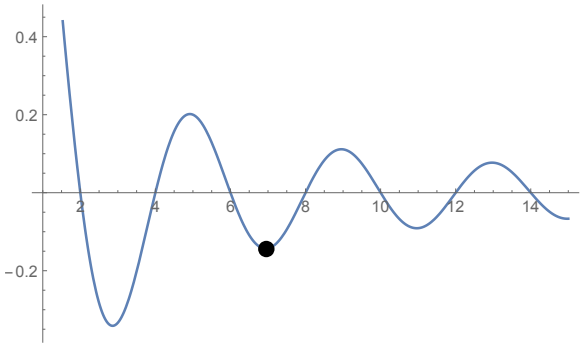

FIGURE 2. Graph of $f(x)$ and our minimum.

How might we check if this is the global minimum? One method could be to add perturbations in our search algorithm and "giggle" it whenever we it found a possible solution. This is our hypothesis behind adding noise to our training data that as the GAN trains every instance of noise will act as a perturbation "giggling" our model a little bit.[7]

[6]The Jensen–Shannon divergence is positive definite and identically 0 when both terms are equal.

[7]This of course is different than our analogy in that this occurs at every iteration of the training process.

5. Experiment

Before we continue any further, it should be noted that for our experiment we opted to use a pix2pix GAN as opposed to the original style of GAN laid out by Goodfellow et.al. Our reason for doing this was because in order for us to get results which we could compare in a meaningful way, within our window of opportunity, we needed to have some control over not just the generated images but also over the generating input. And, if all we had cared about was control over the generated image then a Conditional GAN would have sufficed. However, that extra criteria on the generating input left us with limited options, e.g. a pix2pix, CycleGAN, or DiscoGAN; and, of all of our possible remaining choices, pix2pix rose to the top because our validation image had a known solution which we could compare our results to.

Insofar as the setup of our experiment is concerned, that is quite simple. Using the WikiArt data set, we created two sets of semantic maps(these will act as the domain of our training set), one set with noise and the other without, which can be paired with the original(the range of our training set) image. We then randomly selected an image to use for validation in the training process and used the remaining images to train our pix2pix GAN in randomized minibatches of 16. In the end, we trained two pix2pix models, one using the data set with noise and the other without noise.

To generate the semantic maps, we ran k-means($k = 4$) over the RGB values of the original images, replaced each pixel with the centroid of its classification set, and used a guassian blur kernel to denoise the image. The results of this process reduced the original images to geometrically complicated images with only four colors.[8] It is at this point that we added noise to one of the data sets by replacing the RGB values of approximately $1/4$ of the pixel in every image with uniformly random three-tuples which ranges from $(0, 0, 0)$ to $(255, 255, 255)$. After the two sets of domain images are formed, we "stitched" them to their corresponding range image to form the two data sets which we then trained our two models on.

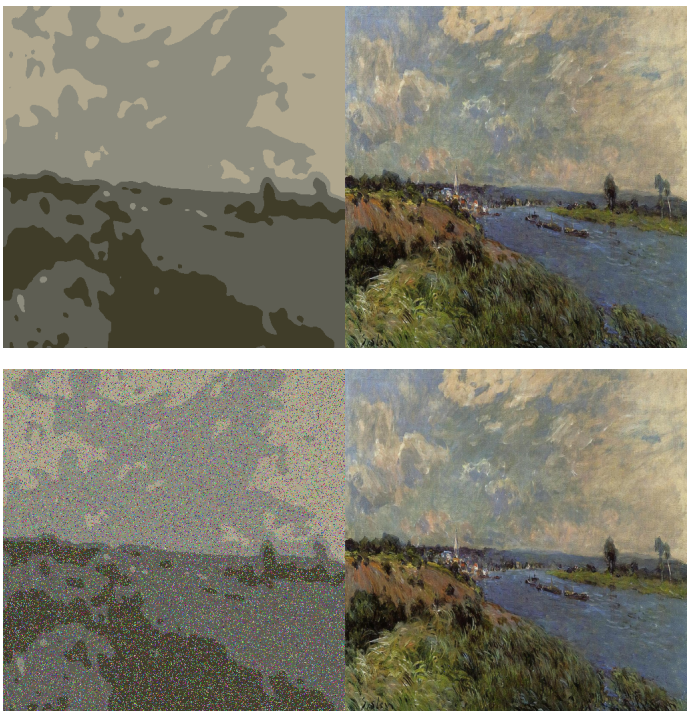

FIGURE 3. Two semantic maps, without noise(top) and with noise(bottom), and their target image

After each generation of the training process, our pix2pix model was applied to the validation image and the resultant image was saved. Because we know the desired target image, it was this image that

[8]These four colors being the the average of the replaced colors

we will use to compare our two models with. We did this by comparing the central tendencies of each of the three images and analyzing the central tendency of image formed by taking the difference of known solution and the generated images.[9] In particular we compared the mean, variance, and mean norm(you can think of this as a measurement of saturation) of the RGB values of the pixels for each of the images generated by the models and the difference-image previously mentioned.

6. Results

At a glance(see figure 2), we can immediately see that qualitatively it appears training with noise led to our model generating an image which was over saturated. As a result, we should expect the mean pixel value and the mean norm to be markedly higher than both our target image and the image generated by the model trained on data without noise.

FIGURE 4. Left to right: target image, generated image with no noise, generated image with noise

Moving now to our quantitative analysis we find the mean and variance of target image and the image generated(see table 1) by the model trained with out noise to be incredibly similar. In fact the difference in the mean pixel value for the two images was only $(0.01, 2.6, 3.2)$. The result of this is that the percent error($\left|\frac{\text{(mean RGB value of target)}-\text{(mean RGB value of generated image)}}{\text{(mean RGB value of target)}}\right|100\%$) for the mean RGB value was only, $(0.00961631, 3.14999, 4.59638)$.

	Target	n.Noise	Targ - n.Noise
Mean	(103.99, 82.54, 69.62)	(103.98, 79.94, 66.42)	(120.98, 111.83, 111.82)
Variance	(2372.71, 1899.24, 2051.54)	(2457.71, 1751.89, 1993.43)	(11903.34, 12446.53, 12797.39)
Mean Norm	151.94	149.48	227.50

TABLE 1. Central tendencies of the RGB values of each pixel - no noise

FIGURE 5. Target image, generated image with no noise, the difference between the two images

[9]It was not until the author went through this process that an error in our setup was found.

Moving our attention now too the model trained with noise(see table 2) we find a different story. The difference in the mean pixel value for the two images was $(-56.39, -47.27, -40.1)$.[10] The result of this is that the percent error for the mean RGB value was $(54.2264, 57.2692, 57.5984)$.

	Target	w.Noise	Targ - w.Noise
Mean	(103.99, 82.54, 69.62)	(160.38, 129.81, 109.72)	(122.63, 103.69, 95.86)
Variance	(2372.71, 1899.24, 2051.54)	(10128.91, 11518.32, 10715.73)	(4565.15, 4263.17, 5614.78)
Mean Norm	151.94	248.83	201.46

TABLE 2. Central Tendencies of the RGB values of each Pixel - With Noise

FIGURE 6. Target Image, Generated Image with Noise, The Difference between the two Images

Moreover, we should note that the variance of the pixel values of the two images in both cases followed this same patter of conformity, with the variance of the image generated from the model trained without noise conforming to that of the target image; and, the variance of image generated from the model trained with noise diverging wildly from that of the target image.

The reader may notice a suspicious lack of comment on the data pertaining to the difference of the target image and generated images. This is because, as the reader saw, the difference between our target image and the image generated by the model trained with noise led to pixel values less than 0 on average. Since such a value doesn't make since, it is nonsensical to consider results from such data. It is the opinion of the author that such a case should have been handled more carefully and considered prior to the start of their work; and, admits that more time is needed to fully understand the process used by the Pillow library to convert Numpy arrays into images. That said, a quick plot of the difference of the means against the mean of the difference-image leads the author to believe a linear scaling method was used, perhaps something like

$$y = \frac{255}{\max_{\hat{x}\in\text{data}} \hat{x} - \min_{\hat{x}\in\text{data}} \hat{x}}(x - \max_{\hat{x}\in\text{data}} \hat{x}).$$

[10]This is where we first identified the error in our setup, what is a negative pixel?

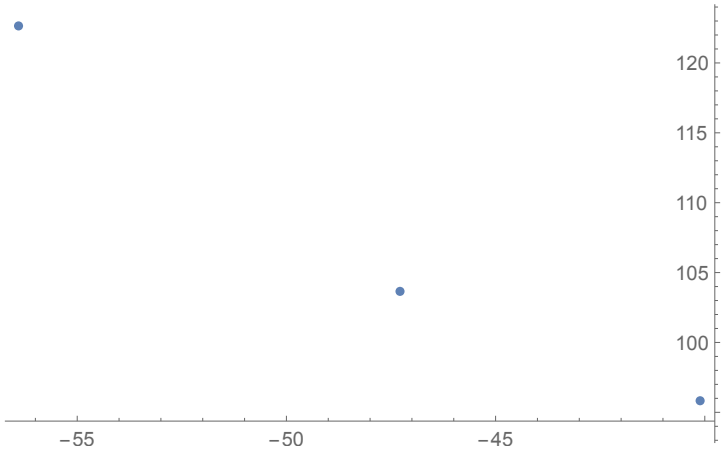

FIGURE 7. Plot of the difference of the means against the mean of the difference-image.

7. Conclusions

In this paper we have shown that noise appears to hinder the training process. However, due to the limited nature of our experiment more investigation is needed before that should be concluded as true. Insofar as the high color saturation in the images generated by our model trained with noise, it is the author's suspicion that this may be because the ubiquity of the noise in our training set is causing our model to weight those colors as more "favorable" while simultaneously identifying how to map the structures of the underlying image. To test this, we propose controlling the RGB values of the replaced pixels to see if the saturated effect changes.

In the future the author plans on developing a method for better handling the difference of the pixel values between the target image and generated images. The author also plans on adding variability to the amount of noise in each image as well as the number of images which have noise added.

8. Appendix

8.1. Code, Data, and Resources for the Reader

- WikiArt Data Set:
  - https://archive.org/details/wikiart-dataset
  - This was used to generate the data in the domain of the GAN, the semantic maps. It was also used as the range of the GAN, the target image.
- Cleaning and generating the data set:
  - https://github.com/MattTucker22689/pix2pix-GAN-Project/tree/main/DataManagement
  - The source code used for cleaning and generating the data used to train the pix2pix-GAN was written by the author of this paper.
- pix2pix:
  - https://github.com/aladdinpersson/Machine-Learning-Collection/tree/master/ML/Pytorch/GANs/Pix2Pix
  - The author of this paper is not the author of the original source code for the pix2pix GAN used in this project.
- Presentation:
  - https://tuckersideascom.files.wordpress.com/2022/12/presentation.pdf?force_download=true
- Validation results and .pth model files from the training process without noise:
  - https://tuckersideascom.files.wordpress.com/2022/12/nonoise.zip?force_download=true
- Validation results and .pth model files from the training process with noise:
  - https://tuckersideascom.files.wordpress.com/2022/12/noise-1.zip?force_download=true
- GitHub repo:
  - https://github.com/MattTucker22689/pix2pix-GAN-Project
  - The following GitHub repo contains all of the code used in this project

8.2. Algorithms for the Reader

Algorithm 1 The following algorithm is of the training process for a GAN using minibatch stochastic gradient descent. *You can find the original stated algorithm in [2] Goodfellow.*

for number of training iterations **do**
  for k steps **do**
    ∘ From noise $p_g(z)$, sample minibatch of n noise samples $\{z^{(1)}, \cdots, z^{(n)}\}$
    ∘ From $p_{data}(x)$, sample minibatch of n noise samples $\{x^{(1)}, \cdots, x^{(n)}\}$
    ∘ Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{n} \sum_{i=1}^{n} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right)\right].$$

  end for
  ∘ From noise $p_g(z)$, sample minibatch of n noise samples $\{z^{(1)}, \cdots, z^{(n)}\}$
  ∘ Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{n} \sum_{i=1}^{n} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

end for
The gradient-based updates can use any standard gradient-based learning rule.

Algorithm 2 The following algorithm is of the Adam optimizer, which was implemented by our code in the "train.py" file.

input: γ (lr), β_1, β_2 (betas), θ_0 (params), $f(\theta)$ (objective) λ (weight decay), amsgrad, maximize
initialize: $m_0 \leftarrow 0$ (first moment), $v_0 \leftarrow 0$ (second moment), $\widehat{v_0}^{\max} \leftarrow 0$

for $t = 1$ to ... **do**
  if *maximize* **then**
    $g_t \leftarrow -\nabla_\theta f_t\left(\theta_{t-1}\right)$
  else
    $g_t \leftarrow \nabla_\theta f_t\left(\theta_{t-1}\right)$
  end if
  if $\lambda \neq 0$ **then**
    $g_t \leftarrow g_t + \lambda\theta_{t-1}$
  end if
  $m_t \leftarrow \beta_1 m_{t-1} + \left(1-\beta_1\right) g_t$
  $v_t \leftarrow \beta_2 v_{t-1} + \left(1-\beta_2\right) g_t^2$
  $\widehat{m_t} \leftarrow m_t / \left(1-\beta_1^t\right)$
  $\widehat{v_t} \leftarrow v_t / \left(1-\beta_2^t\right)$
  if *amsgrad* **then**
    $\widehat{v_t}$ max $\leftarrow \max\left(\widehat{v_t}\,\max, \widehat{v_t}\right)$
    $\theta_t \leftarrow \theta_{t-1} - \gamma\widehat{m_t} / \left(\sqrt{\widehat{v_t}^{\max}} + \epsilon\right)$
  else
    $\theta_t \leftarrow \theta_{t-1} - \gamma\widehat{m_t} / \left(\sqrt{\widehat{v_t}} + \epsilon\right)$
  end if
end for
return θ_t

9. Reference

[1] Foster, D. (2019). Generative deep learning: teaching machines to paint, write, compose, and play. O'reilly.
[2] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., et al. (2014). Generative Adversarial Nets. https://proceedings.neurips.cc/paper/2014/file/5ca3e9b122f61f8f06494c97b1afccf3-Paper.pdf
[3] Hany, J., & Walters, G. (2019). Hands-On Generative Adversarial Networks with PyTorch 1.x. Packt Publishing Ltd.
[4] Isola, P., Zhu, J.-Y., Zhou, T., Efros, A., & Research, B. (2018). Image-to-Image Translation with Conditional Adversarial Networks. https://arxiv.org/pdf/1611.07004.pdf
[5] Kalin, J. (2018). Generative Adversarial Networks Cookbook. Packt Publishing Ltd.
[6] Maheshkar, S. (2021, September 10). What Is Cross Entropy Loss? A Tutorial With Code. W & B. https://wandb.ai/sauravmaheshkar/cross-entropy/reports/What-Is-Cross-Entropy-Loss-A-Tutorial-With-Code--VmlldzoxMDA5NTMx. Accessed 28 November 2022
[7] Maschler, M., Solan, E., & Zamir, S. (2013). Game Theory. Cambridge University Press.
[8] Perarnau, G. (2017, March 17). Fantastic GANs and where to find them. guimperarnau.com. https://guimperarnau.com/blog/2017/03/Fantastic-GANs-and-where-to-find-them. Accessed 28 November 2022
[9] Robert Duncan Luce, & Raiffa, H. (1989). Games and decisions introduction and critical survey. Dover Publications.
[10] Zhu, J.-Y., Park, T., Isola, P., Efros, A., & Research, B. (2020). Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks. https://arxiv.org/pdf/1703.10593.pdf

Department of Mathematics, Wichita State University, Wichita, KS, U.S.A.
Email address: mtucker@math.wichita.edu