Usage of GANs in simulations: a brief overview

Andrej Leban ANDREJ_LEBAN@BERKELEY.EDU
Department of Statistics

Abstract

In the following work, I provide a brief overview of the use of generative adversarial networks in simulations, focusing on Monte Carlo simulations on problems from Physics. After introducing the method, I showcase an elementary example: the 2D Ising model of ferromagnetism. The model was trained on states generated via Metropolis' algorithm. The results were found to be in agreement with the exact simulation both qualitatively and quantitatively via the distributions of a statistical property - average magnetization. The example is followed by showcases of use in three different fields: high-energy particle simulations, gravitational lensing maps in cosmology, and the integration of stochastic differential equations, pertinent to quantitative finance. All these share the same basic architecture with the example implemented, illustrating the flexibility of the DCGAN architecture.

1. Preface

While most of us have likely heard about the usage of *GAN*s for somewhat nefarious purposes, such as the creation of *deepfakes*, the application that has been of most interest to me for quite a while is their application in drastically speeding-up computation in physics.

Since I was quite familiar with the latter and not entirely sure how the former manages to achieve this, this project presented a unique opportunity to look into a new, extremely interesting subject through a familiar lens.

2. An extremely brief introduction to GANs

GANs stand for *Generative Adversarial Networks* and were introduced by Goodfellow et al. (2014). Their salient feature is the *adversarial* property: they consist of two models which compete against each other. The *generator* G tries to generate synthetic data as close to the input dataset's true distribution without being given the true data. On the other hand, the *discriminator* D is trying to "catch" it, being given a copy of the true data. Both are trained via backpropagation from the discriminator's classification error: the generator is trying to maximize it and, conversely, the discriminator to minimize it. Hence they compete in a zero-sum game.

Formally, we can express the training as a *minimax game* for the objective functional V (Goodfellow et al., 2014):

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

where $D(x)$ is a single number outputted by the discriminator representing the probability of the data being genuine. Thus minimizing $\mathbb{E}\left[log(1-D(G(z)))\right]$ represents the generator trying to fool the discriminator. In the example presented in sec. 3, as is most often the

case, the z input is noise distributed as a standard normal and amounts to sampling the generator's transformed space at random.

Since we are dealing with unsupervised learning, the interesting question is: when do we stop training? It is known from game theory that all zero-sum games have a *Nash equilibrium* (Langr and Bok, 2019), or conversely, an optimum for convex-concave zero-sum games is guaranteed by the *minimax theorem*. In this specific case, this amounts to the discriminator's accuracy being the same as a random coin toss, which means that the generator's output is indistinguishable from real data, i.e. $p_{gen} = p_{data}$.

While all this does hold, the practical obstacle is that the game of training the network is not guaranteed to be convex, making it potentially exceedingly slow to converge to an equilibrium.

3. A concrete illustration: the Ising model

The Ising model is a relatively simple model of ferromagnetism in statistical physics that attempts to explain it via the interaction of atomic spins. In the often-studied 2D example, the *Hamiltonian* of the system is given by a "linear" and "interaction" part:

$$H(s) = -H \sum_i s_i \; - \; J \sum_{(i,j) \text{ neighborss}} s_i s_j, \tag{2}$$

where H is the magnetic field density and J is the *pairwise interaction strength*. The first term is zero in the absence of an external magnetic field, which is the case I'll be examining in this illustration.

Each *spin configuration* s has the probability given by the *Boltzmann distribution*:

$$\mathbb{P}(s|T) = \frac{1}{Z} \, e^{-H(s)/kT}, \tag{3}$$

where Z is the *partition function*.

The equilibrium state is traditionally found using the Metropolis algorithm (or other MCMC techniques), with the above proposal density and the action at each step being a flip of a single spin. In contrast to the uses of MCMC in, for example, hierarchical Bayesian models, the "model" here is solely the that the spins interact on a lattice as above. The sampled distribution is given by the laws of physics!

Besides the interaction strength, a crucial variable is the temperature. When using dimension-less units ($k_B = 1, J = 1$), the *Curie temperature* T_C is approximately 2.27 units. This is the point above which ferromagnetic matter starts to lose permanent magnetization, which should, in our model, correspond to the decay of contiguous domains of parallel spins.

An example of a spin configuration below T_C is presented below:

3.1 The architecture

This illustrative model is inspired by Liu et al. (2017), who used *convolutional* and *transpose-convolutional* neural networks for the generator and discriminator elements. This enabled them to side-step the simulation part and essentially "learn" the correct equilibrium Ising configurations for a with the temperature T as a parameter:

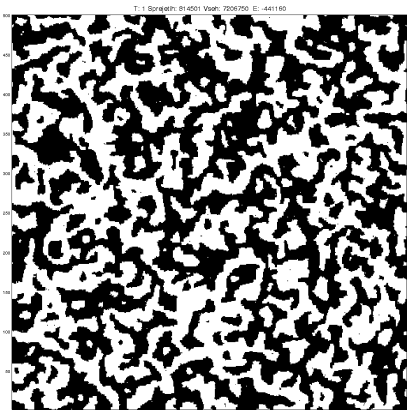


Figure 1: The Ising model at a sub-critical temperature (Leban, 2015)

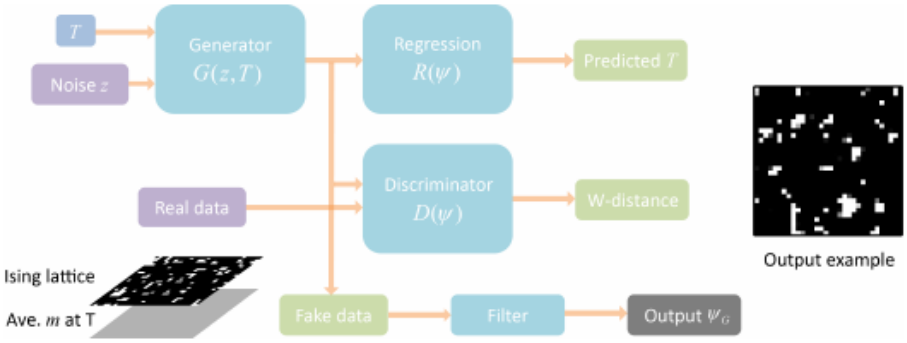


Figure 2: The architecture in Liu et al. (2017)

They, in turn, took their inspiration from Radford et al. (2016) - the structure of the convolutional networks remained practically identical. Since the latter used it to generate images of everyday objects, this might seem an unusual choice at first. However, the Ising states *are* in a sense but black-and-white (1-channel) images, parametrized by a couple of parameters and with the true data distribution known: the Boltzmann distribution. Given that the network architecture is able to generate much more complex image categories with non-closed form data distributions, it should be able to generate images with an explicit data distribution.

The generator thus consists of 5 transpose-convolutional layers with *batch normalization* and $ReLU$ activation functions, with a $tanh$ final output activation mapping it into $\{-1, 1\}$. The discriminator, conversely, uses 5 convolutional layers with the reverse kernel structure, *leaky* $ReLU$ activation, and, as discussed below, *spectral normalization* in lieu of the original batch normalization. The optimization is performed with the *Adam* algorithm.

Since I believe the audience to be generally familiar with the above architecture, I shall briefly illustrate how it maps to the picture obtained from physics:

While the Hamiltonian (eq. 2) is itself given by a form of discrete convolution (with the Manhattan-nearest neighbors kernel), the architecture does not proceed from such first-

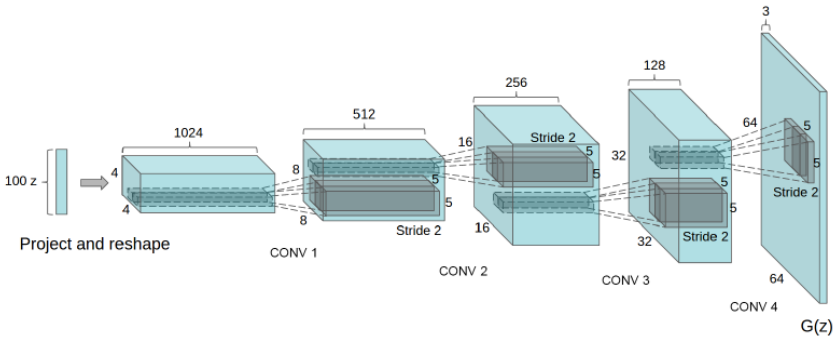


Figure 3: The generator architecture from Radford et al. (2016); it is unchanged (down to the dimensions) in Liu et al. (2017) and in my implementation.

principle considerations. Instead, the discriminator performs the encoding into a nonlinear *transformed space* via a series of discrete convolutions followed by non-linear activation functions, as in Radford et al. (2016).

The input state $|\psi\rangle$ is described as a row-major concatenation of the component spins of length $N \cdot N$, i.e. $|-1, 1, -1, ...\rangle$. Each application of a convolutional kernel down-samples the state into a state $|\psi'\rangle$, with much-reduced dimensionality. At each of the five layers, N_k kernels are applied. The resulting $|\psi'_k\rangle$ are again concatenated, the activation function is applied, and are fed into the next layer. In practice, this gives us a non-linear transformation (Liu et al., 2017):

$$\psi^f = \mathbf{P}(\psi^0) = \tilde{P}_L \circ \tilde{P}_{L-1} \circ \ldots \tilde{P}_1 \, \psi^0 \tag{4}$$

as a composition of the non-linear transformations $\tilde{P}_L$ performed by each layer. In our example, we encode an 64×64 image into a 100-long feature space, where each element represents a highly transformed representation of (potentially) the whole input.

The generator's role is, of course, reversed: it samples this feature space with a normally-distributed noise input z and generates 64×64 Ising states.

Since the architecture in (Liu et al., 2017) takes the temperature as a free parameter, the authors introduce an additional feature that helps disambiguate Ising states that are similar in the transformed feature space but stem from different temperatures. They use the average (absolute) magnetization per spin:

$$\langle s \rangle = \frac{1}{N} \sum_i^N |\psi_i|, \tag{5}$$

which is just the mean of the absolute value of the state. This distribution of magnetization across states for a given temperature is also the quantitative metric considered in section 3.2.

The generator's loss from eq. 1 is therefore augmented with regularized discrepancies from the generated magnetization and the temperature inferred by the regression module

(fig. 2). The latter predicts a temperature from the generated state - the generator is thus additionally penalized for temperature-inappropriate states.

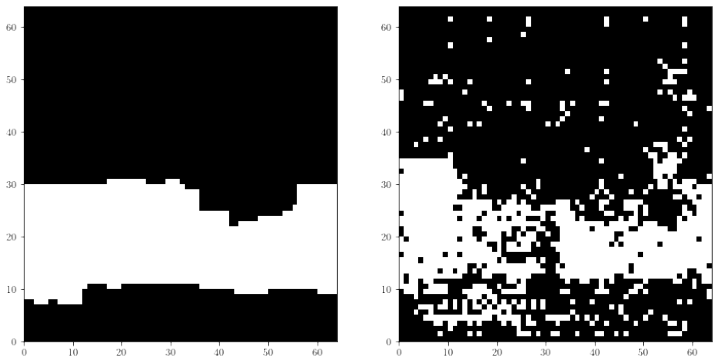
I decided to somewhat simplify the architecture in my implementation. Unlike Liu et al. (2017), I kept the temperature fixed and trained separate networks for each temperature. Thus I did not train the *Regression* network from fig. 2. I did, however, keep the magnetization as a feature (and penalty term).

In the final implementation, I also used two ideas from the best-performing reference implementation of the above neural net in Liang et al. (2019), such as replacing batch normalization in the discriminator with *spectral normalization* (Miyato et al., 2018) and replacing the Wasserstein distance loss with the *hinge loss*. The former uses normalization of each layer's weights by their largest singular value to ensure Lipschitz-continuity of the discriminator's function $D(x)$. This is similar to regularizing the gradients and works to prevent *mode collapse* - the feedback loop where only a narrow set of outputs are deemed acceptable by the discriminator and, consequently, fed back to the generator.

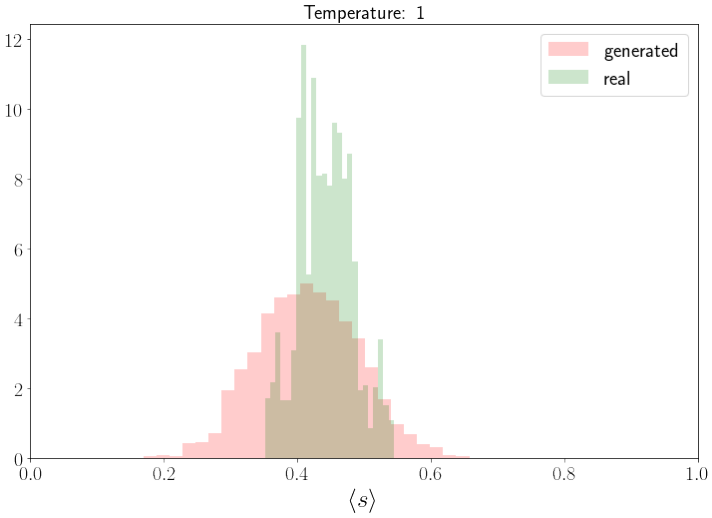
3.2 The results

The code for both the Monte Carlo simulator and the GAN implementation is available at Leban (2021). The following results were obtained after training the network for five epochs:

- $T = 1$**:**

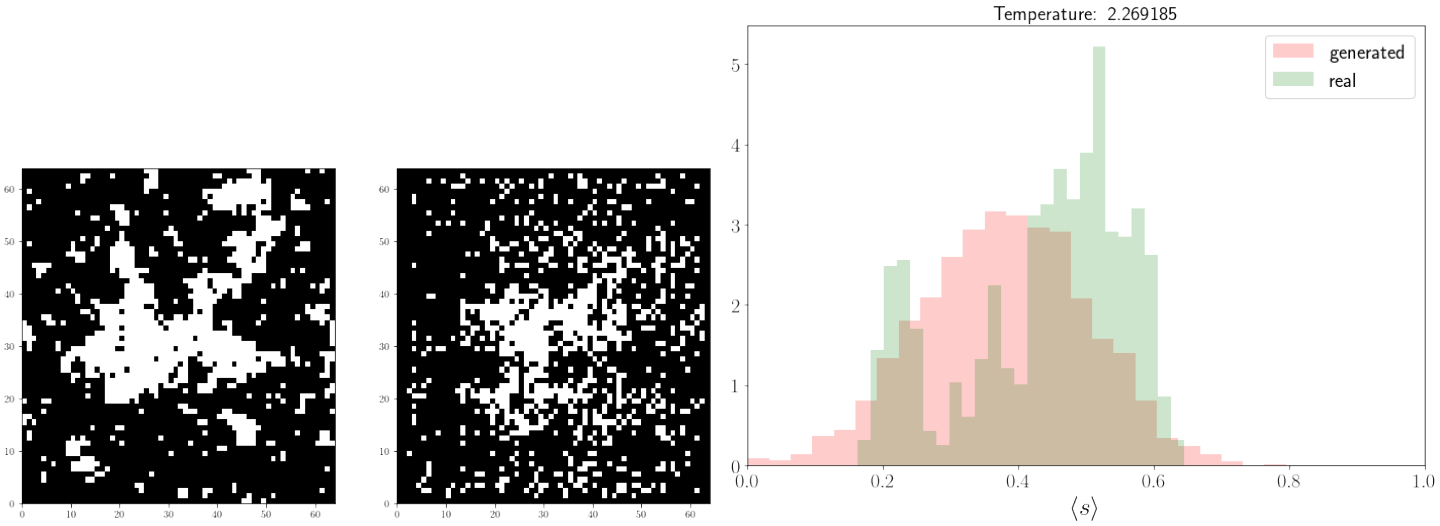


(a) Example states: *left*: MCMC, *right*: GAN



(b) Magnetization distributions

- $T = T_c$**:**



(a) Example states: *left*: MCMC, *right*: GAN

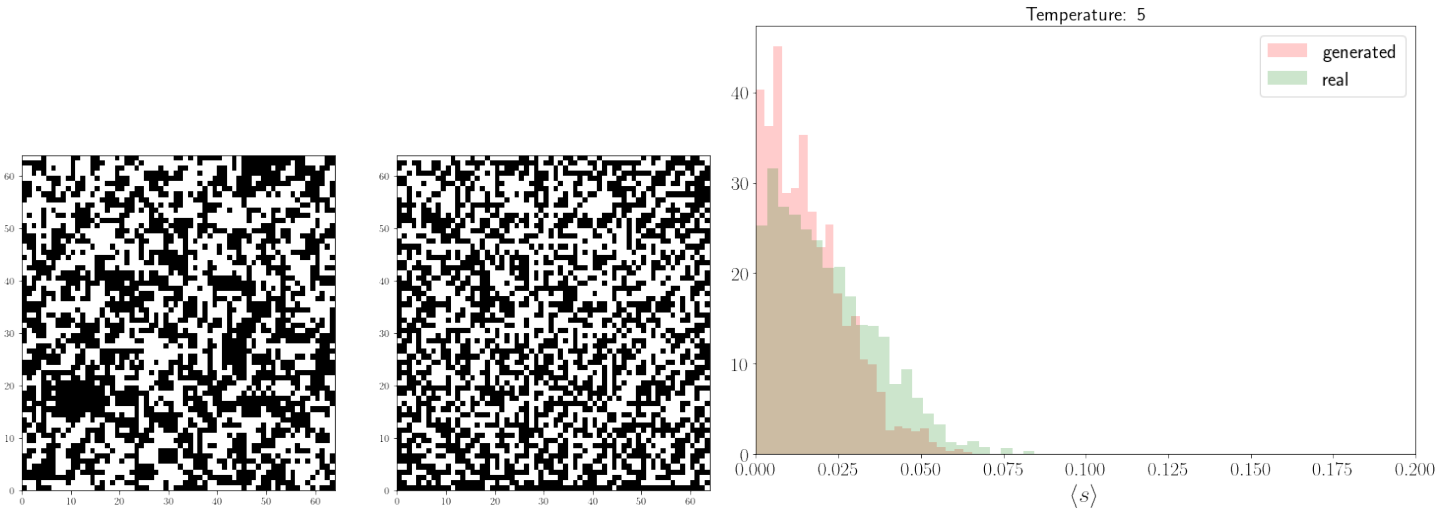
(b) Magnetization distributions

The increase in temperature leads to the decay of the magnetic domains, which is also reflected in the increased variance of the average (absolute) magnetization. I am inclined to think the bi-modality of the real distribution is mostly an artifact of not "burning-up" enough iterations in the MCMC integrator. In this usage, however, we wish to generate a diverse array of states as the training input for the GAN to escape *mode collapse*. Therefore I erred on the side of including this run with some not-quite-equilibrium states, since, in this case, tweaking the MC procedure resulted in a chain oscillating around more-or-less a single configuration. In other words, I sacrificed accuracy on the final distribution for the qualitative accuracy of the generated states. In any case, the generated data distribution interpolates between the peaks of the training one, which is a desired quality, since it's indicative of the network's ability to generate unseen samples.

- $T = 5$**:**

The results seem to have, qualitatively and quantitatively, captured the essence of the different regimes. In the qualitative sense, however, one can see both a bit too much regularity, as well as too much "noise", which is directly analogous to the "uncanny valley" effect that is apparent when generating, for example, human faces (Radford et al., 2016). It also needs to be said that both the input data and the network configuration could still be further tweaked for better results.

As for the practical aspects, once trained, the generator can generate thousands of different Ising states in less than a second, while, even on this toy example, reaching equilibrium for higher temperatures can take minutes via MCMC, not to speak of the question of *chain mixing*. This becomes even more obvious when one moves to higher dimensions, where the time requirements of a trained generator increase negligibly compared to those of MCMC procedures.



(a) Example states: *left*: MCMC, *right*: GAN

(b) Magnetization distributions

4. A brief overview of some other examples of use

4.1 High-energy physics

Vallecorsa (2018) illustrates the use of conditional GANs (Mirza and Osindero, 2014) as a potential replacement for complex Monte Carlo simulations of particle transport at the Large Hadron Collider at CERN, specifically the transit of particles through a 3D array of *calorimeters*. This transit registers in the array as an energy shower. The results from the sensors can be, much like the example (3), thought of as an image - in this case, three-dimensional. Each energy shower profile, therefore, corresponds to a "state".

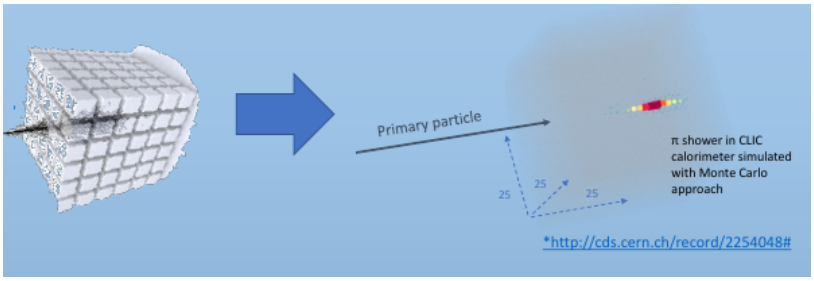


Figure 7: A schematic illustration of a particle passing through the calorimeter array (source)

The architecture of the network is largely simply an "up-dimensioned" version of the original DCGAN (Radford et al., 2016). An additional feature, somewhat similar to the regression module in Liu et al. (2017), is that the discriminator also classifies the particle and predicts its energy. Similar to the use of magnetization in section 3, this is fed back to the generator. The results for the energy dissipation profiles of particles of differing mass/energy are shown in figure 8.

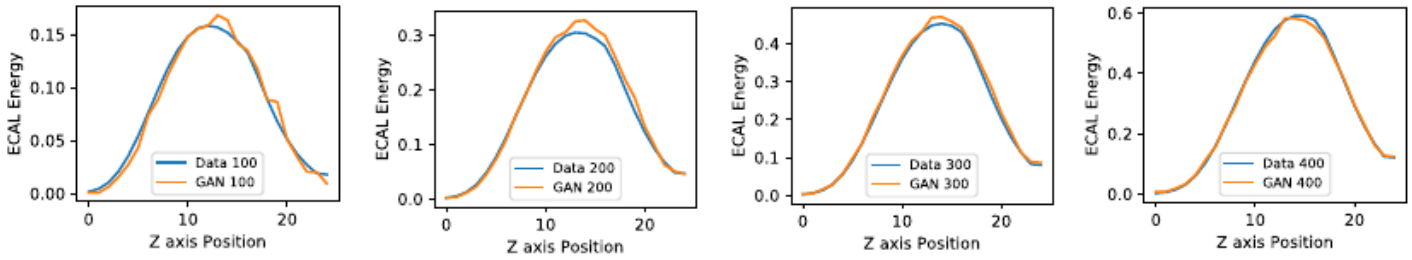


Figure 8: Generated *longitudinal shower shapes* at different particle energies compared to measured ones. The x-axis is the longitudinal direction, i.e. the distributions show the dissipation *rate* of energy in the calorimeter (Vallecorsa, 2018).

Since similar Monte-Carlo computations take up as much as 50% of CERN's high-performance computing capacity (Vallecorsa, 2018), their potential replacement with such models would drastically speed up the rate of research.

4.2 Cosmology

Mustafa et al. (2019) used the same architecture as both the examples above to generate *weak lensing convergence maps*. These are maps of the luminosity of background sources as acted upon a massive object in the foreground, resulting in the shearing and magnification of light. Traditionally, they are generated by directly simulating an *N-body problem* ($\mathcal{O}(N^2)$) from first principles (i.e. general relativity) - this example, therefore, does not seek to replace a Monte Carlo simulation.

The network architecture and parameters remained largely unchanged from Radford et al. (2016), with the minor difference of adding a fully connected layer in front of the generator and halving the number of convolutional kernels used. The real and generated maps are presented in fig. 9.

As is customary, the authors also picked physical quantities whose distributions should match those of the data. They found a good match on all of them, such as *pixel intensity* presented in fig. 10.

4.3 Finance

As an example, van Rhijn et al. (2021) examined the use of GANs in solving *Stochastic differential equations (SDEs)*, which are the backbone of pricing complex financial instruments. While closed-form formulas exist for the most basic ("vanilla") of products (if one accepts the Gaussian nature of log-returns), things quickly move into Monte Carlo integration, and from there, simulation, especially when one starts considering human action in pricing.

The authors used GANs to learn the conditional distribution of the solution S to a Brownian motion SDE:

$$S_{t+\Delta t} \,|\, S_t = G_\theta(Z, \Delta_t, S_t), \tag{6}$$

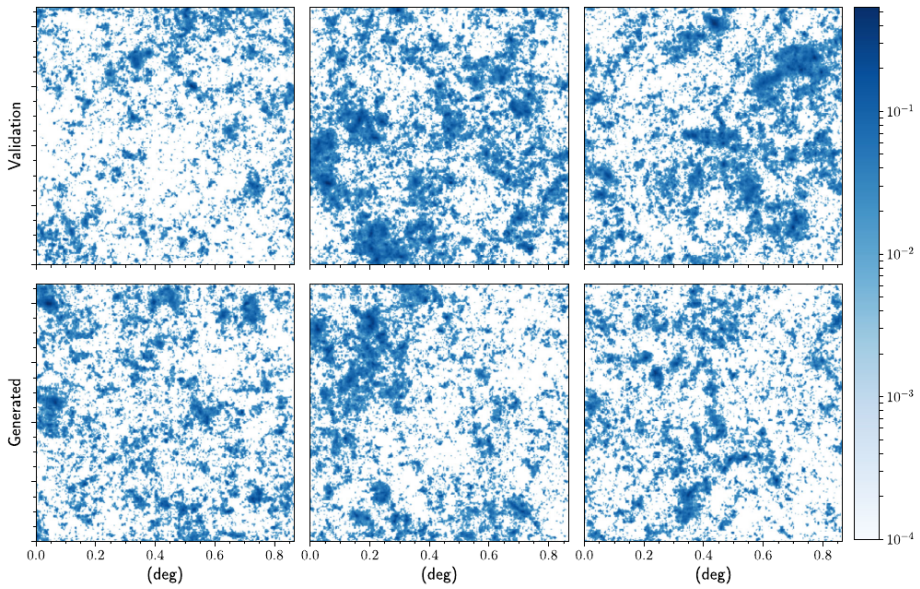


Figure 9: Examples of generated and validation (i.e. unseen) gravitational lensing convergence maps (Mustafa et al., 2019).

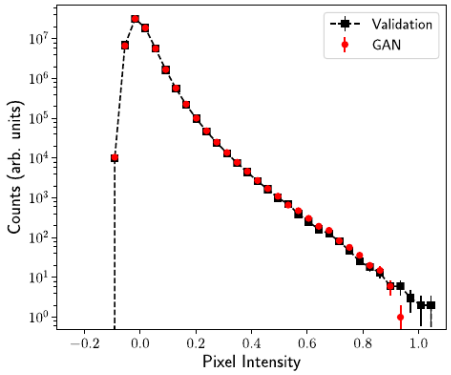
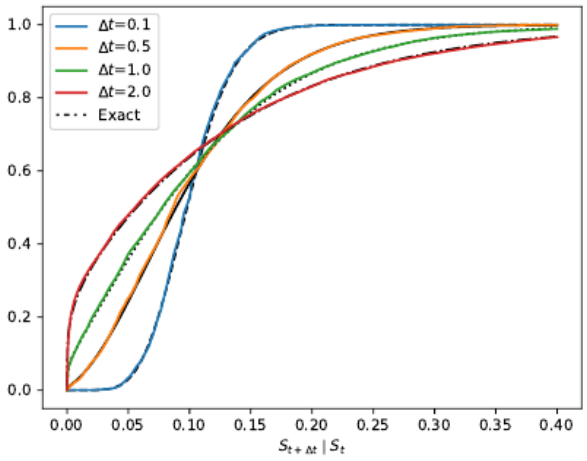


Figure 10: Distributional match in pixel intensity between the generated and real data (Mustafa et al., 2019).

where Z is the standard normal (note the *Markov property* of the SDE) and G is the generator's output. This was then used to correctly propagate the solution forward. Moreover, they modified the architecture so that the generator received additional training input in order to satisfy complementary required relations for a *strong solution* of the SDE, which is path-wise unique. This lead them to call this implementation of a *conditional GAN* (Mirza and Osindero, 2014) a "supervised GAN".



(b) Supervised GAN

Figure 11: The match between the conditional distributions (eq. 6) of the exact solution and the ones obtained via the "supervised GAN" scheme (van Rhijn et al., 2021)

5. Conclusion

In this work, I hoped to illustrate some of the uses of generative adversarial networks in (mainly) physics simulations. Most of the examples were Monte Carlo simulations and motivated by computational complexity, not modeling, concerns. The "models" underneath were often exact in e.g., the proposal distribution.

If we compare GANs to the wider MCMC models (such as hierarchical models), however, we seem to be trading the posterior distribution obtained by the latter for the generator's distribution, which approximates the true data distribution. In comparison to the former, the latter is non-parametric and, additionally, robust to missing data (or capable of interpolation) - it generates examples not seen in the training sample, as verified statistically in e.g. Mustafa et al. (2019).

However, the example of *mode collapse* has me wondering where the exact limits of this interpolation ability are. Especially in first-principle simulations, such as the ones from physics shown in this work, any valid configuration is, theoretically, obtainable by simulation, including previously unseen ones. I am left with the question of whether one would arbitrarily exclude outliers if one were to replace simulation with a generative network wholesale, without making sure the training set is wholly representative of any possible data configuration.

References

Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative Adversarial Networks. 2014.

Jakub Langr and Vladimir Bok. *GANs in action: deep learning with generative adversarial networks*. Manning, 2019.

Andrej Leban. Modelska analiza 1::9: Metropolisov algoritem. `https://andleb1.files.wordpress.com/2021/12/andrej_leban_109.pdf`, 2015. Computational Physics assignment, Dept. of Mathematics and Physics, University of Ljubljana.

Andrej Leban. Solving the Ising model using GANs. `https://github.com/andleb/ganIsing`, 12 2021.

Shujia Liang, Fangze Liu, and Tianyi Liu. Deep Convolutional Generative Adversarial Network for Simulating the 2D Ising Model. `http://cs230.stanford.edu/projects_winter_2019/reports/15813032.pdf`, 2019. CS 230 final project, Stanford University.

Zhaocheng Liu, Sean P. Rodrigues, and Wenshan Cai. Simulating the Ising Model with a Deep Convolutional Generative Adversarial Network. 2017.

Mehdi Mirza and Simon Osindero. Conditional Generative Adversarial Nets, 2014.

Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral Normalization for Generative Adversarial Networks. 2018.

Mustafa Mustafa, Deborah Bard, Wahid Bhimji, Zarija Lukić, Rami Al-Rfou, and Jan M Kratochvil. CosmoGAN: creating high-fidelity weak lensing convergence maps using Generative Adversarial Networks. *Computational Astrophysics and Cosmology*, 6(1):1, 2019.

Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks, 2016.

S. Vallecorsa. Generative models for fast simulation. *Journal of Physics: Conference Series*, 1085:022005, sep 2018. doi: 10.1088/1742-6596/1085/2/022005. URL `https://doi.org/10.1088/1742-6596/1085/2/022005`.

Jorino van Rhijn, Cornelis W. Oosterlee, Lech A. Grzelak, and Shuaiqiang Liu. Monte Carlo Simulation of SDEs using GANs. 2021.