Validating Deep Learning Simulations

Anton Björklund
firstname.lastname@helsinki.fi
Department of Computer Science
University of Helsinki
Helsinki, Finland

1 INTRODUCTION

Simulators are used in many science disciplines. These commonly require intensive calculations and / or lots of small scale interactions, which makes the simulators really slow [14]. For other problems a lack knowledge or resources prohibits building simulators in the first place. With the recent boom in machine learning a relevant question is if machine learning could provide approximative solutions to these problems?

The machine learning method that has gotten the most attention recent years is deep learning, which usually refers to neural networks with multiple hidden layers. Neural networks could be particularly well suited for simulation tasks, since they are universal function approximators. In other words you can approximate any function to any desired precision with a neural network given enough capacity [5]. Furthermore, neural networks are by nature very parallel and can utilise accelerators, such as GPU:s, for speed boosts of several orders of magnitude [14].

The big drawback of using deep learning is the lack of insight in what the model is actually doing. This means that there is no way of analytically verifying that the model is correct, the validation has to happen statistically. This is the topic of this review, exploring how to validate deep learning simulations.

The validation methods gathered in this review are from both simulation and non-simulation sources. This means that many of the methods are designed for a specific type of model, type of data, or domain. But usually they can be adopted to work in other settings.

On a high-level, the training of a deep learning simulators falls into one of two categories, supervised or generative. For supervised training you start by measuring the inputs and outputs of the system, either the real world or a scientific simulator. Then you train the neural network to infer the output from the input. Generative models on the other hand start from little to no information and tries to generate realistic outputs.

In a supervised setting you have access to the ground truth, which makes validation easier. So in this review will emphasise the more difficult case, generative models. Note that validation methods for generative models also works on supervised models, having access to the truth doesn't change anything.

Generative models are useful when you cannot capture enough information about the initial situation, or the process is inherently non-deterministic, like in high-energy physics [1]. There are multiple architectures for generative models to choose from, like variational auto-encoders [9]. But recently GAN:s [2] (Generative Adversarial Networks) have gotten a lot of attention due to their ability to generate very realistic samples [7].

This review is structured as follows. In Sec. 2 is a short overview of how GAN:s work, since they are state-of-the-art for generative models. That provides an insight into the problems a generative model can face and serves as something the validation methods can relate to. The main focus of the review is in Sec. 3 where the validation methods are presented. Finally in Sec. 4 is discussed how domain knowledge can be utilised even before the validation step for increased quality.

2 GENERATIVE ADVERSARIAL NETWORKS

The lack of ground truth makes training of generative models difficult. Instead of relying on the data to provide feedback for the training you have to do it yourself. However it can be quite difficult to design a loss function that accurately measures the quality of the generated samples. Generative adversarial networks (GAN) solve this by introducing a second neural network.

GAN:s can be described as a game between two neural networks, a generator and a discriminator. The task of the generator is to generate samples from (mostly) random noise. The discriminator is trained to classify real and generated samples, through supervised training. The result of the discriminator is in turn used to train the generator to generate more real-like samples. This game ends at a Nash equilibrium, when the generator cannot improve further because the discriminator cannot separate real samples from fake samples [2].

Formally this can be expressed with the optimisation of the (value-) function in Eq. 1. Here D is the discriminator, G the generator, x the real samples, and z the (random) latent vector used as input to the generator. Note that this definition is *not* limited to any specific kind of domain, data, or model. This enables the GAN architecture to be applied to *any* generative task.

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

The main objective for a generative model is to create a diverse set of high-quality samples. For simulations this translates into creating realistic samples with the same variety as the training samples (with similar probability distributions). Fortunately the Nash equilibrium of GAN:s *should* provide just that, but there are problems that can arise during training which stops the GAN from reaching that point.

The three main types of GAN failures are; non-convergence when the training never stabilises, model collapse when the generator only produces a few different samples, and memory-GAN when the generator exactly replicates the training data. The causes of these failure states are not always known and can be anything from insufficient data to the generator and discriminator getting out of balance.

Since the generator updates are dependent on the discriminator gradients this is a likely source of problems. Indeed even the first GAN paper [2] offered an alternative definition of the loss function for the generator which avoids saturation and vanishing gradients. More recent findings show that adding gradient normalisation to the discriminator helps with convergence [12].

There have been a lot of recent advances in GAN:s that are worth mentioning [11, 13, 15, 18] and also some from the wider deep learning research [3, 6, 8] that are advantageous to use with GAN:s. But since this review focuses on validation they won't be discussed further. The failure modes, on the other hand, is something the validations in the next section should be able to detect.

3 VALIDATION

The biggest challenge with deep learning is that it is almost impossible to get any useful information from the raw values that the model has learned (this is known as the black box problem). With simulators this is further compounded by the fact that we don't know what the "best" approximations are (otherwise we wouldn't need deep learning). This means that you have to rely on empirical methods for validating your simulator. Fortunately there is lots of approaches that can be, and have been, used.

If you have access to the ground truth, as is the case with supervised training, then you can use normal error and accuracy measures. You can even prioritise the correctness for different situations, outcomes, and variables differently. Often these preferences can be formulated as a loss function that you then you can use to directly train your model with.

The more difficult, and interesting, case is when you don't have the truth to compare with, such as with generative models. An obvious way of validating is to manually inspect the results. The best outcome would be if the simulator is able to trick human experts [10]. Manual inspection is very time consuming, compared to automated methods, so fewer samples tend to be examined. Combining humans with few samples can lead to very subjective and biased conclusions, which makes comparison between models unreliable. But manual inspection is still needed, since if a deep learning simulator fails an expert examinations then it is obviously not up to the task.

Automated validations are based on calculations. This means a lot of samples can be inspected and the results are not subjective, but they might still be biased! The bias comes from potential assumptions and heuristics used in the calculations. This is why it is preferable to use many, if not all the methods presented below. Another advantage with automated validations is that the results might be numerical values, which enables comparisons between different simulators (or the same simulator at different stages of training).

If the validation calculates some quality measure of the samples, then as the simulator improves, so should the quality *score*. However, the goal of a simulator is to generate realistic data and realistic data might not always correspond to having a high quality *score*. An alternative is to do a comparison between real and generated data, then you get a *distance*. The more the generated samples resembles real data the smaller the *distance*. Both *scores* and *distances* can be used to measure and compare simulators, but since *distances* rely on real data instead of some heuristic they *might* be less biased.

The most simple validation is to measure the distance between the distribution of real data and generated data for each variable. Distributions are used for all the distance measures presented in this review because we are interested in the simulator as a whole, not just one single sample. Distributions also show if the simulator is skewed in any systematic way (remember the lack of diversity from the previous section).

Comparing variables on their own is however usually insufficient, since that doesn't account for dependencies, e.g. the colour of a single pixel doesn't matter without seeing the larger picture. To remedy this, more complicated, higher level, features are used, often with the help of domain knowledge. An example of this is when Paganini et al. [14] calculated the geometrical and physical shapes of particle showers for their particle simulator (and compared the distribution of those).

In most domains there are some rules or constraints that the real world follows. The same constraints must of course be followed by deep learning simulators. If these constraints can be calculated then they can be used to validate models. One of the most important constraints in physical systems is conservation of energy, which Rasp et al. [16] noted happening even without explicitly adding that to their model.

One of the reasons the GAN architecture uses a discriminator is because the problem of determining what is a good sample is in itself a difficult problem. Even though the discriminator is trained to rate samples it cannot be used for validation, since it is a part of the simulator and has the same biases as the generator. Using a neural network in the validation process is, however, the idea behind the *Inception Score* [18]. The Inception Score uses an existing (unrelated) classifier to calculate some qualities about the samples.

The Inception Score is designed with a specific multi-class classifier in mind, and in many domains nothing equivalent exists. It is also not a distance so no comparison to real data is made. Thus an improvement has been suggested, the *Fréchet Inception Distance* (FID) [4]. FID compares the datasets by calculating the Fréchet distance between the distributions of means and covariates at the last hidden layer in an "existing" classifier (which doesn't need to be multi-class).

A similar idea has been used with deep learning particle simulations [14]. First two classifiers were trained, one only on real data and one only on generated data. Then they were evaluated with generated data or real data respectively. For both evaluation to yield high accuracy the generated samples needs to be diverse and of high-quality [19].

Using another black box to validate a black box seems counterintuitive at first, but this is just another way of constructing high level features. When high level features where mentioned above they were designed by a human, usually with some domain knowledge. This limits them to features that person is able to think of, which might not cover everything needed for realistic data. By using a black box we construct new, but unknown, high level features.

The only thing we can say about those features is that they are somewhat relevant for the domain in question, since they are used by the classifier (especially FID). But the use of high level features is still justified, even if they are unknown, because if generated data really resembles real data then the generated data should resemble real data also for those high level features. Note that I'm not arguing for replacing transparent validation methods, but rather augment them (you can never have too many validation options).

What classifier-based validation fails to uncover is if the model has memorised the training data, only generating samples from the training data [11]. This would of course yield short distances, but the model would still not be a good simulator. Paganini et al. [14] solved this by manually comparing generated samples to the closest real samples (according to sample distances, e.g. euclidean, not distribution distances).

Whenever you train a deep learning simulator you do it with a goal in mind. That goal could also be used to verify that the simulator accomplishes its task. For example if the goal is to improve medical classifiers by generating more cases of rare diseases [10, 17] then the goal metric would be to set aside a validation set (before training) that would be used to measure how much an classifier improves when trained on both real and generated data, compared to the same classifier trained only on real data.

4 UTILISING DOMAIN KNOWLEDGE

Domain knowledge could of course be utilised earlier in the process, yielding better simulators. This process starts with the model design, you want yo select the right kind of deep learning model, e.g. convolutional neural networks for images. This selection is dependent on the data, but requires more deep learning knowledge than domain knowledge.

Since training a simulator end-to-end is difficult, the training can be guided by giving the model a domain specific auxiliary task. The auxiliary task helps the training focus on the underlying structure, even before the simulator has gotten good enough to need it. An auxiliary task for GAN:s could be to add a supervised classification task to the discriminator, e.g. classifying the particle types in the real data [1].

A related idea is semi-supervised learning [18]. Here the simulator is given additional information, e.g. a category, that it tries to adhere to. If the model is a GAN then the same information is given to both networks for both real and generated data. This guides the discriminator to learn the connection between that information and the final results. A big advantage with semi-supervised simulators is that you can influence what kind of samples you want it to generate [17].

Sometimes there is knowledge about how to best accomplish a part of the simulation, either exactly or heuristically. If this part is especially difficult or crucial then you might not want the deep learning to handle it. The solution is to build a pipeline consisting of both learned and predefined steps. Lau et. al. [10] successfully used this technique to generate scars on healthy patients. They used a three step process; first they selected the area with a GAN, then filled it with new tissue based on a heuristic, and finally they used another GAN to add realistic details to and around the new tissue.

5 CONCLUSIONS

Deep learning is under very active development. This can especially be seen in the progress of generative models and the applications to other scientific fields. When it comes to deep learning simulators we have seen a lot of promising works, but some of them also provide convincing results.

The lack of transparency with deep learning also affects simulators. From a scientific point of view the difficulty of extracting the learned knowledge is really unfortunate since it limits the applicability of the models. As a consequence deep learning simulators are only being used to augment datasets or data collection. That doesn't mean that deep learning simulators are useless, they can still provide massive speed-ups or help when real samples are rare or expensive to acquire.

The use of deep learning simulators comes down to whether you can trust the simulators. The easiest way to build trust is through rigorous and diverse testing, and this review have explored some of the methods that have been used to validate models.

REFERENCES

[1] Luke de Oliveira, Michela Paganini, and Benjamin Nachman. 2017. Learning Particle Physics by Example: Location-Aware Generative Adversarial Networks for Physics Synthesis. *Computing and Software for Big Science* 1, 1 (29 Sep 2017), 4. https://doi.org/10.1007/s41781-017-0004-6

[2] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative Adversarial Nets. In *Advances in Neural Information Processing Systems 27*, Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger (Eds.). Curran Associates, Inc., 2672–2680. http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf

[3] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep Residual Learning for Image Recognition. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. http://openaccess.thecvf.com/content_cvpr_2016/html/He_Deep_Residual_Learning_CVPR_2016_paper.html

[4] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. 2017. GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium. In *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.). Curran Associates, Inc., 6626–6637. http://papers.nips.cc/paper/7240-gans-trained-by-a-two-time-scale-update-rule-converge-to-a-local-nash-equilibrium.pdf

[5] Kurt Hornik. 1991. Approximation capabilities of multilayer feedforward networks. *Neural Networks* 4, 2 (1991), 251 – 257. https://doi.org/10.1016/0893-6080(91)90009-T

[6] Sergey Ioffe and Christian Szegedy. 2015. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *International Conference on Machine Learning*. 448–456. https://arxiv.org/abs/1502.03167

[7] Tero Karras, Samuli Laine, and Timo Aila. 2018. A Style-Based Generator Architecture for Generative Adversarial Networks. *CoRR* abs/1812.04948 (2018). https://arxiv.org/abs/1812.04948

[8] Diederik P. Kingma and Jimmy Ba. 2014. Adam: A Method for Stochastic Optimization. arXiv:cs.LG/1412.6980 https://arxiv.org/abs/1412.6980

[9] Diederik P Kingma and Max Welling. 2013. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114* (2013). https://arxiv.org/abs/1312.6114

[10] Felix Lau, Tom Hendriks, Jesse Lieman-Sifry, Sean Sall, and Dan Golden. 2018. Scargan: chained generative adversarial networks to simulate pathological tissue on cardiovascular MR scans. In *Deep Learning in Medical Image Analysis and Multimodal Learning for Clinical Decision Support*. Springer, 343–350. https://arxiv.org/abs/1808.04500

[11] Mario Lucic, Karol Kurach, Marcin Michalski, Sylvain Gelly, and Olivier Bousquet. 2018. Are GANs Created Equal? A Large-Scale Study. In *Advances in Neural Information Processing Systems 31*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman,

N. Cesa-Bianchi, and R. Garnett (Eds.). Curran Associates, Inc., 700–709. http://papers.nips.cc/paper/7350-are-gans-created-equal-a-large-scale-study.pdf

[12] Lars Mescheder, Andreas Geiger, and Sebastian Nowozin. 2018. Which Training Methods for GANs do actually Converge?. In *International Conference on Machine Learning*. 3478–3487. https://arxiv.org/abs/1801.04406v4

[13] Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. 2018. Spectral normalization for generative adversarial networks. *arXiv preprint arXiv:1802.05957* (2018). https://arxiv.org/abs/1802.05957v1

[14] Michela Paganini, Luke de Oliveira, and Benjamin Nachman. 2018. Accelerating science with generative adversarial networks: an application to 3D particle showers in multilayer calorimeters. *Physical review letters* 120, 4 (2018), 042003. https://arxiv.org/abs/1705.02355

[15] Alec Radford, Luke Metz, and Soumith Chintala. 2015. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434* (2015). https://arxiv.org/abs/1511.06434

[16] Stephan Rasp, Michael S. Pritchard, and Pierre Gentine. 2018. Deep learning to represent subgrid processes in climate models. *Proceedings of the National Academy of Sciences* 115, 39 (2018), 9684–9689. https://doi.org/10.1073/pnas.1810286115

[17] Hojjat Salehinejad, Shahrokh Valaee, Tim Dowdell, Errol Colak, and Joseph Barfett. 2018. Generalization of deep neural networks for chest pathology classification in x-rays using generative adversarial networks. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 990–994. https://arxiv.org/abs/1712.01636

[18] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. 2016. Improved techniques for training gans. In *Advances in neural information processing systems*. 2234–2242. http://papers.nips.cc/paper/6124-improved-techniques-for-training-gans

[19] Konstantin Shmelkov, Cordelia Schmid, and Karteek Alahari. 2018. How good is my GAN?. In *The European Conference on Computer Vision (ECCV)*. http://openaccess.thecvf.com/content_ECCV_2018/html/Konstantin_Shmelkov_How_good_is_ECCV_2018_paper.html