swissQuant
Mastering Complexity

Mathematical Challenge October 2018

Generative Adversarial Networks

References

[1] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, in *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'14 (MIT Press, Cambridge, MA, USA, 2014) pp. 2672–2680, arXiv:1406.2661 .

[2] D. Pfau and O. Vinyals, in *NIPS workshop 2016*, NIPS'16 (2016) arXiv:1610.01945 .

[3] C. Finn, P. Christiano, P. Abbeel, and S. Levine, in *NIPS workshop 2016*, NIPS'16 (2016) arXiv:1611.03852 .

[4] M. Arjovsky, S. Chintala, and L. Bottou, (2017), arXiv:1701.07875 .

[5] I. Goodfellow, (2016), arXiv:1701.00160 .

[6] C. Papadimitriou, in *Proceedings of the Thirty-third Annual ACM Symposium on Theory of Computing*, STOC '01 (ACM, New York, NY, USA, 2001) pp. 749–753.

[7] L. Mescheder, S. Nowozin, and A. Geiger, in *Advances in Neural Information Processing Systems 30*, edited by I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Curran Associates, Inc., 2017) pp. 1825–1835, arXiv:1705.10461 .

[8] L. Mescheder, A. Geiger, and S. Nowozin, (2018), arXiv:1801.04406 .

[9] V. Nagarajan and J. Z. Kolter, (2017), arXiv:1706.04156 .

Description

In this mathematical challenge we take a look at Generative Adversarial Networks (GANs) [1]. GANs belong to the class of *generative* probabilistic models (such as Boltzmann machines or Gaussian Mixture Models), which are designed to learn a probability distribution function of possibly very high dimensional data. So far GANs have been applied mainly in computer vision and natural language processing but many other fields stand to benefit from their further development. For example there is strong interest in applying them for certain control, learning, and planning tasks which require an accurate internal model of the environment in order to select optimal actions [2, 3].

A key feature of a GAN (reflected by its name) is that it can be naturally viewed as a two-player game between two adversaries. One of the players (the discriminator) is trained towards being able to tell whether a given sample was drawn from the data distribution or whether it was produced by the other player (the generator). Conversely, the generator is trained towards being able to fool the discriminator (i.e. make it assign probability 1/2 to inputs from either class). The optimal solution of this game is a Nash equilibrium, i.e. a situation where neither player can gain by deviating from their current strategy. GANs are at the intersection between supervised, unsupervised and reinforcement learning and have therefore attracted a lot of attention in these machine learning communities. From the mathematical perspective GANs also present some intriguing connections to transport theory [4] and game theory [5]. The latter is particularly interesting as the problem of finding a Nash equilibrium has been proposed as one of the most important open problems in complexity theory [6].

Despite the early success of GANs in computer vision, it is not yet clear how, when and why they actually work. In this short challenge we will first present a gentle introduction to the basic GAN theory and then briefly explore one central open problem – convergence to equilibrium. A set of questions and references to some of the literature are included to stimulate further investigation.

1 Ideal GAN training

Goodfellow *et al.* [1] formulate the GAN training as the following minimax zero-sum game:

$$\begin{aligned}&\min_G \max_D V(D,G)\\ &V(D,G) = \mathbb{E}_{x\sim p_{\text{data}}} \log D(x) + \mathbb{E}_{z\sim p_z} \log(1 - D(G(z))). \end{aligned} \tag{1}$$

Here $D : \mathcal{X} \to [0,1]$ is the discriminator function, which assigns a probability of coming from the data distribution to a given sample $x \in \mathcal{X}$ and $G : \mathcal{Z} \to \mathcal{X}$ is the generator function to be learned, which maps input data from $\mathcal{Z} \subset \mathbb{R}^l$ onto sample space $\mathcal{X} \subset \mathbb{R}^d$ (note that $l \geq d$). To gain some intuition about the cost function V let us fix the generator distribution and seek the optimal discriminator strategy. We have

$$\mathbb{E}_{x\sim p_{\text{data}}} \log D(x) + \mathbb{E}_{z\in p_z} \log(1 - D(G(z))) \tag{2a}$$

$$= \int_{\mathcal{X}} p_{\text{data}}(x) \log D(x) dx + \int_{\mathcal{Z}} p_z(z) \log(1 - D(G(z))) dz \tag{2b}$$

$$= \int_{\mathcal{X}} \Big[p_{\text{data}}(x) \log(D(x)) + p_{\text{model}}(x) \log(1 - D(x))\Big] dx. \tag{2c}$$

Taking a functional derivative with respect to $D(x)$ immediately yields the optimal discriminator

$$D^*(x) = \frac{p_{\text{data}}(x)}{p_{\text{data}}(x) + p_{\text{model}}(x)}. \tag{3}$$

As a corollary, the GAN minimax game is equivalent to finding

$$\min_G C(G), \tag{4}$$

where

$$C(G) = \mathbb{E}_{x\sim p_{\text{data}}}\left[\log \frac{p_{\text{data}}(x)}{p_{\text{data}}(x) + p_{\text{model}}(x)}\right] + \mathbb{E}_{x\sim p_{\text{model}}}\left[\log \frac{p_{\text{model}}(x)}{p_{\text{data}}(x) + p_{\text{model}}(x)}\right]. \tag{5}$$

♦ Q1: Show that the global optimum is achieved when $p_{\text{data}} = p_{\text{model}}$. [Hint: relate C to the Jensen-Shannon divergence $D_{JS}(p,q) = \{D_{KL}(p||(p+q)/2) + D_{KL}(q||(p+q)/2)\}/2$, where $D_{KL}(p||p) = \mathbb{E}_{x\sim p}[\log(p(x)/q(x))]$ is the Kullback-Leibler divergence.]

---

2 Real GAN training

The discussion so far has assumed that the optimization is carried out directly in function space. This is clearly not a tractable problem in general. Instead what Goodfellow *et al.* [1] proposed is to parametrize the discriminator and generator, $D(x) \to D(x, \theta_D)$ and $G(z) \to G(z, \theta_G)$, using (deep) neural networks (i.e. the parameters θ_D and θ_G represent the weights and biases of two separate feed-forward neural networks) and then use a (stochastic) gradient-based method (such as back-propagation) to solve the minimax problem in an iterative fashion. For example one simple algorithm for solving zero-sum two-player games is the simultaneous gradient descent/ascent algorithm:

Algorithm 1 SimGrad

```
1: while not converged do
2:     $v_D \leftarrow \nabla_{\theta_D} V(\theta_D, \theta_G)$
3:     $v_G \leftarrow \nabla_{\theta_G} V(\theta_D, \theta_G)$
4:     $\theta_D \leftarrow \theta_D + \eta_D v_D$
5:     $\theta_G \leftarrow \theta_G - \eta_G v_G$
```

3 Minimax and gradient methods

From a theoretical perspective, the above steps are anything but innocuous. In fact now that optimization takes place in parameter space rather than function space, the nice results about existence of and (global) convergence to an equilibrium no longer hold [5]. It is

presently unclear if and under what conditions the above gradient based algorithm can reach an equilibrium. To illustrate this, let us consider a simple toy example [5]:
Consider a minimax game with the value function $V(x, y) = xy$ for $x, y \in [-1, 1]$. Suppose the minimizing player controls x while the maximizing player controls y.

♦ Q2: Does this game have an equilibrium and if so can it be found by the above SimGrad method? [Hint: consider an infinitesimal gradient step.]

As this simple example illustrates gradient based methods applied to GAN may not always converge to an equilibrium, even when the latter exists. A recent work [7, 8], investigates the stability of SimGrad and proposes a regularized gradient algorithm called Consensus Optimization. In the following we briefly explore a variation thereof [9].

Algorithm 2 Variant of Consensus Optimization

```
1: while not converged do
2:     $v_D \leftarrow \nabla_{\theta_D}\left(V(\theta_D, \theta_G) - \gamma||\nabla_{\theta_G} V(\theta_D, \theta_G)||^2\right)$
3:     $v_G \leftarrow \nabla_{\theta_G}\left(V(\theta_D, \theta_G) + \gamma||\nabla_{\theta_D} V(\theta_D, \theta_G)||^2\right)$
4:     $\theta_D \leftarrow \theta_D + \eta_D v_D$
5:     $\theta_G \leftarrow \theta_G - \eta_G v_G$
```

♦ Q3: Show that using this algorithm the equilibrium of the above toy model can indeed be found. [For local stability proofs for the general case see [7, 9, 8]]. How would you modify the back-propagation algorithm to include this regularization?

We look forward to your opinions and insights.

Best Quant Regards,

swissQuant Group Leadership Team