IFT 6756 - Lecture 11
(Wasserstein Generative Adversarial Nets)

This version of the notes has not yet been thoroughly checked. Please report any bugs to the scribes or instructor.

Scribes **Instructor:** Gauthier Gidel
Winter 2021: [François David, Justine Pepin, Bharath Govindaraju]

1 Summary

In the previous lectures, we covered different divergence minimization perspectives. The standard Generative Adversarial Network formulation correspond to minimizing the KL divergence.

$$\min_{p_g} \max_{D} \mathop{\mathbb{E}}_{x \sim p_{data}} [log(D(x))] + \mathop{\mathbb{E}}_{x' \sim p_g} [log(1 - D(x'))] \tag{1}$$

Since the optimal disctiminator can be described as :

$$D(x) = \frac{p_{data}}{p_{data} + p_g}$$

Then we can re-write the maximization step over the discriminator as the Jenson-Shannon divergence between the generated distribution and the original distribution:

$$\min_{p_g} JS(p_g || p_{data}) - log(4) \tag{2}$$

Where the Jenson-Shannon divergence is :

$$JS(p \parallel q) = KL\left(p \parallel \frac{p+q}{2}\right) + KL\left(q \parallel \frac{p+q}{2}\right)$$

$$KL(p \parallel q) = \int_x log\left(\frac{p(x)}{q(x)}\right) p(x) dx$$

The paper Arjovsky et al. [1] is motivated by the comparisons of "distance" between the two distribution. It proposes a metric distance to compare the generated distribution and the data distribution, the Wasserstein distance:

$$W(p, q) = \inf_{\gamma \in \Pi(p,q)} \mathop{\mathbb{E}}_{(x,y) \sim \gamma} [||x - y||]$$

which is also called the "Earth Mover Distance". The Wasserstein distance have some appropriate properties for GAN training compared to the Jenson-Shannon distance. Those properties will be described in further sections. An intuitive way to understand the motivation of this metric is to consider the optimal transport problem (next section).

2 Optimal Transport

The optimal transport gives a framework for comparing two different measures by assigning a cost to transporting one measure to another. In 1781, a french mathematician, named Gaspard Monge, formulated the following problem. Assuming we have a certain amount of soil that we can extract from different locations, we need to transport that soil to multiple locations to construct a remblais (Villani [6]). However, the cost of transporting the soil is expensive and we need to minimize that cost. Therefore, we need to map the extraction sites to the construction sites as efficiently as possible. Initialy, Monge's assumption was that the transport cost is the product of the mass times the distance.

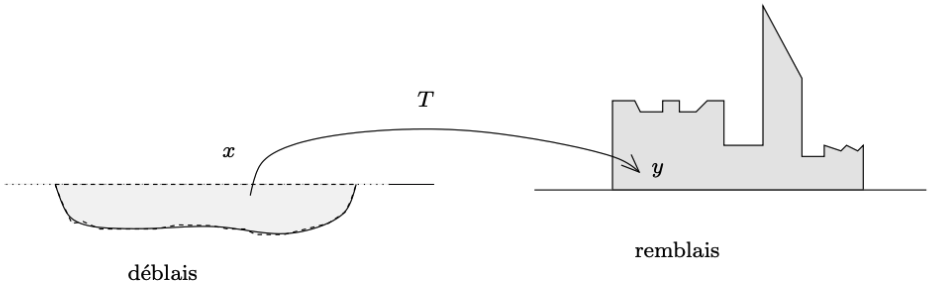


Fig. 3.1. Monge's problem of déblais and remblais

Figure 1: Illustration from Villani [6]. It illustrates the problem of the Monge's formulation where we want to transport mass from déblais to construct a designed remlais while minimizing the cost of transportation.

2.1 Discrete Case Formulation

Let's formulate the problem in the discrete case, the intuitions can be generalized to the continuous case.
We have the initial distribution:

$$\alpha = \sum_{i=1}^{n} p_i \delta_{x_i}$$

We have the target distribution:

$$\beta = \sum_{j=1}^{m} q_i \delta_{y_j}$$

Where p_i correspond to the mass assosiated with point i, and q_j is the mass needed at destination j. Also δ_{x_i} and δ_{y_j} correspond to the diract delta function of their respective variables (representing the discrete points). Also note that, in some cases, $m \neq n$ since the number of points where we need soil can differ from the number of extraction sites.

We want to learn the mapping: T: $\{x_i\} \longrightarrow \{y_j\}$ such that all the construction sites receives the amount necessary for proper construction.

$$q_j = \sum_{i:T(x_i)=y_j} p_i \tag{3}$$

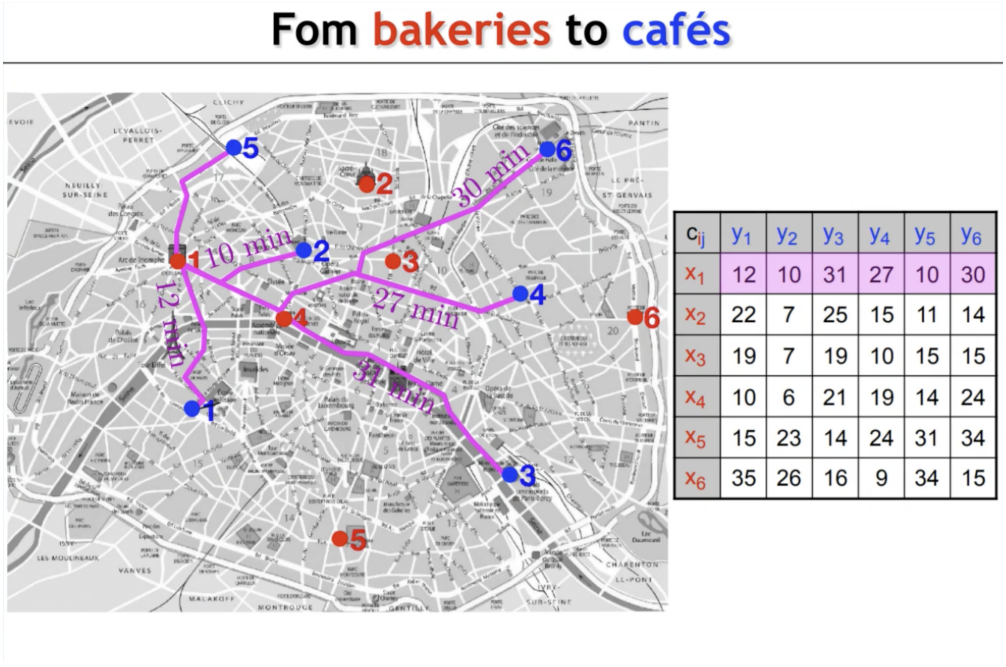
Among all the admissible mappings T, we want the one that minimizes the cost:

$$\min_{T} \sum_{i=1}^{n} c(x_i, T(x_i)) \tag{4}$$

where $c(x_i, T(x_i))$ is the cost of transporting x_i to $T(x_i)$. This is a non-convex optimization problem which is not easy to solve numerically Peyré and Cuturi [4]. What makes the initial mapping matrix are the binary constraints, that the sum of each row and collumn should equal a vector of ones.

2.2 Bakeries-Cafés Example

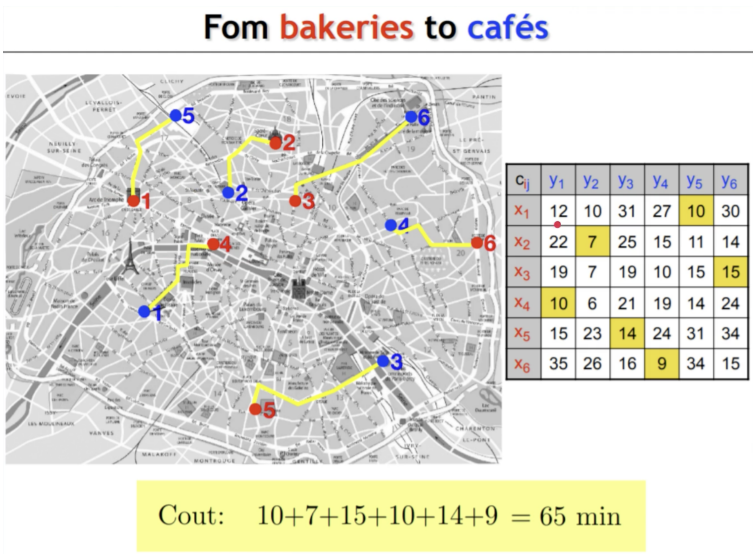
In this example provided by Peyré and Cuturi [4], we want to find the mapping between bakeries to cafés. The idea is that we want to transport pastries to the corresponding cafés, which will then be sold to consumers. The variable y in the following mapping matrix correspond to cafés and the variable x correspond to the bakeries. The values in the mapping matrix are the cost of transportation between x_i and y_j. In this context, the cost is considered to be the time it takes to transport the pastries from point A to point B. As we can see below, there are as many one-to-one mapping as there are permutation of the data, which leads to a computationally expensive problem if we want to solve it naively.



c_{ij}	y_1	y_2	y_3	y_4	y_5	y_6
x_1	12	10	31	27	10	30
x_2	22	7	25	15	11	14
x_3	19	7	19	10	15	15
x_4	10	6	21	19	14	24
x_5	15	23	14	24	31	34
x_6	35	26	16	9	34	15

Figure 2: Illustration from Peyré and Cuturi [4]. It illustrates an application of Monge's formulation (in the discrete case) where we want to map bakeries to cafés with the smallest transportation cost possible.

This problem can be very challenging since the number of solutions that satisfy the problem's constraints corresponds to all the possible permutations of cafés and bakeries. The number of solutions scales exponentially with the size of the matrix, finding the optimal solution is not tractable when n is large. Just as an example, in a case where we want to map 6 bakeries to 6 cafés, there are 720 possibles mapping matrices. The illustration below shows one solution (that is not optimal) where we find a mapping between all the bakeries and cafés.



c_{ij}	y_1	y_2	y_3	y_4	y_5	y_6
x_1	12	10	31	27	**10**	30
x_2	22	**7**	25	15	11	14
x_3	19	7	19	10	15	**15**
x_4	**10**	6	21	19	14	24
x_5	15	23	**14**	24	31	34
x_6	35	26	16	**9**	34	15

Figure 3: Illustration from Peyré and Cuturi [4]. It illustrates one of the solutions to the above situation where we want to map bakeries to cafés. This solution has a total cost of 65 minutes.

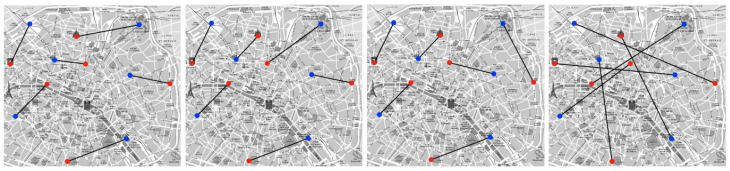


Figure 4: Illustration from Peyré and Cuturi [4]. It illustrates 4 of the 720 possible solutions to the above situation where we want to map bakeries to cafés.

2.3 Spliting Mass

In this problem, we may want to split the pastries prepared by a bakery and ship those splits to different destinations. In this situation, our mapping matrix becomes a coupling matrix that determine the pastries' allocation. This coupling matrix is called P. If we summed all the elements in the rows, we should obtain the original vector X, which correspond to the vector of pastries that can be transported from the bakeries. Similarly, if we summed the columns of the matrix, we should obtain the vector Y, which corresponds to the demand of pastries at each café. Obviously, all its element must be non-negative. More formally, it has the following properties:

$$\begin{aligned} P \in & \mathbb{R}_+^{n \times m} \\ P1 = & X \\ P^T 1 = & Y \end{aligned}$$

Now our objective function becomes:

$$\min_P \sum_{i,j} P_{i,j} C_{i,j} \tag{5}$$

where $P_{i,j}$ corresponds to the pastries transported from x_i to y_j and $C_{i,j}$ is the transportation cost from x_i to y_j. The genius of this reformulation is that we transform the discrete set of mapping matrices, which was very large (n!), by a set of continuous matrices which is infinite but simpler to deal with. By allowing to split mass, the new set of coupling matrix are bistochasic matrices, which are convexes [4]. This new problem is now a convex-linear problem and it scales linearly with the size of the matrix. We can apply algorithms like the simplex, as described by [5], to a convex-linear problem and solve it fairly easily. If x and y are uniform distribution of $U(1/n)$, we obtain a relaxation that has the same solution as in 2.2, exept that the complexity of the problem is much lower since it does not scale exponentially with the number of cafés and bakeries.

2.4 Connection to Wasserstein Distance

As we can see, the Wasserstein Distance is a specific instance of this problem :

$$W(p,q) = \inf_{\gamma \in \Pi(p,q)} \mathop{\mathbb{E}}_{(x,y)\sim\gamma} [||x-y||]$$

$$\begin{aligned} P \in & \mathbb{R}_+^{n \times m} \\ P1 = & X \\ P^T 1 = & Y \end{aligned} \qquad \min_P \sum_{i,j} P_{i,j} C_{i,j}$$

where $\gamma \in \Pi(p, q)$ is a generalization of the coupling in the continuous case and $C_{i,j}$ correspond to the L1 distance between the points x_i and $y_j := ||x_i - y_j||_1$. The $P_{i,j}$ correspond to the expectation in the Wasserstein equation. The infimum correspond to the greatest number that is smaller or equal to all other members in that set, it can be refered to as the greatest lower bound. In the context of GANs, p would be the data distribution and q would be the generated distribution. The smaller this optimal transportation cost is, the closer the distribution are (in that perspective).

3 Motivation for Wasserstein Distance

For gradient descent based learning methods desirable properties of the loss function are that it be continuous and differentiable during the learning phase. Through a example below we can see how Wasserstein(W) distance has better properties than Jenson-Shannon(JS) and KL divergence for simple probability distribution.

Example 1: Consider Z $\sim$ U([0,1]), latent distribution uniform in [0,1]. Let $g_\theta(z) = (\theta, z)$ be the generator that maps z to (θ, z). Let $P_{target} \sim$ U[(0,Z)] target distribution which is a uniform distribution over Y-axis with $x = 0$. Let Q_θ be the data distribution of the generator parallel to target distribution for any θ.
We can see that,

- W(P,Q) = $\inf_{\gamma \in \Pi(P,Q)} \mathbb{E}_{(x,y)\sim\gamma}[||x - y||] = |\theta|$
- JS(P,Q) = $\frac{1}{2}$(KL(P||$\frac{P+Q}{2}$) + KL(Q||$\frac{P+Q}{2}$))= $\begin{cases} \log(2) & \text{if } \theta \neq 0 \\ 0 & \text{if } \theta = 0 \end{cases}$

Proof:
KL(p||q) = $\int_x p(x) log(\frac{p(x)}{q(x)}) dx$

If $\exists x \ni$q(x)=0 andp(x) $\neq$ 0 then $log(\frac{p}{q}) \to \infty$

For $\theta \neq 0$,

- $q_\theta(j) \sim$ (0,z) , KL($p_d||q_\theta$) = $\int_{x=(0,0)}^{(0,1)} log(\frac{1}{q_\theta(x)}) dx$ for P is a uniform distribution in [0,1]
  $\forall \theta \neq 0, \forall a \in [0,1]q_\theta(0, a) = 0$, thus KL $\to \infty$

  KL(q$_\theta||p_d$) = $\int_{x=(\theta,0)}^{(\theta,1)} log(\frac{1}{p(x)}) dx$ for Q is a uniform distribution in [0,1]
  for $x \neq 0$, $p_d(x) = 0$ thus, KL $\to \infty$

- JS(P||Q) = $\frac{1}{2}$ (KL(P||$\frac{P+Q}{2}$) +KL(Q||$\frac{P+Q}{2}$)) = $\frac{1}{2}$($\int_{x=0} log(\frac{1}{(1+0)/2})$ dx)+($\int_{x=\theta} log(\frac{1}{(0+1)/2})$ dx) = log(2)

KL and JS divergence leads to saturation as seen above.

Figure 5 illustrates the above case where JS is locally saturated to max value of log(2) and the gradient is 0. Whereas, Wasserstein distance captures how close θ is to 0 and we get useful gradients almost everywhere (except when $\theta = 0$) as Wasserstein measure cannot saturate and converges to a linear function.

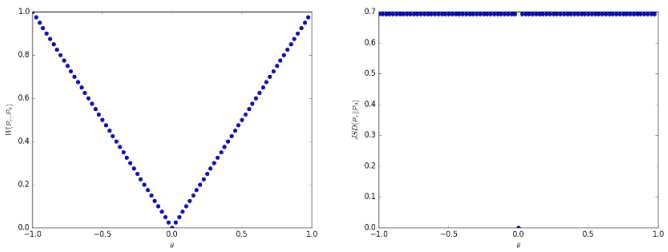


Figure 5: Wasserstein distance(left), JS divergence(right)

In the following theorem from [1] we see under what conditions is Wasserstein distance, $W(\mathbb{P}_r, \mathbb{P}_\theta)$ continuous.

Theorem 1. *Let $\mathbb{P}_r$ be a fixed distribution over $\mathcal{X}$. Let Z be a random variable (e.g Gaussian) over another space $\mathcal{Z}$. Let $g : \mathcal{Z} \times \mathbb{R}^d \to \mathcal{X}$ be a function, that will be denoted $g_\theta(z)$ with z the first coordinate and θ the second. Let $\mathbb{P}_\theta$ denote the distribution of $g_\theta(Z)$. Then,*

1. *If g is continuous in θ, so is $W(\mathbb{P}_r, \mathbb{P}_\theta)$.*
2. *If g is locally Lipschitz and satisfies regularity assumption , then $W(\mathbb{P}_r, \mathbb{P}_\theta)$ is continuous everywhere, and differentiable almost everywhere.*
3. *Statements 1-2 are false for the Jensen-Shannon divergence $JS(\mathbb{P}_r, \mathbb{P}_\theta)$ and all the KLs.*

Regularity assumption from [1] in Appendix A

If the generator is continuous then the Wasserstein distance is continuous with respect to θ and differentiable almost everywhere. This is not true for JS/KL divergence. If we compute the Wasserstein distance between the real data distribution and the generated data distribution, it's a function of θ which is differentiable almost everywhere and could be minimized. In the case of JS divergence, it saturates in some cases and we can get vanishing gradients.

4 Connections between Wasserstein and Jenson-Shannon

Let us consider the dual formulation to see how close the two measures are.

Max in GAN is a divergence

$$JS(p_g||p_d) = \max_D \mathop{\mathbb{E}}_{x\sim p_{data}}[log(D(x)] + \mathop{\mathbb{E}}_{x^{'}\sim p_g}[log(1 - D(x^{'})] - log(4)$$

Wasserstein can be written as a max

$$W(p_g||p_d) = \max_{||F||_L \leq 1} \underbrace{\mathop{\mathbb{E}}_{x\sim p_d}[F(x)] - \mathop{\mathbb{E}}_{x^{'}\sim p_g}[F(x^{'})]}_{\text{optimization problem}}$$

For Wasserstein we see that the max is constrained where discriminator is 1-Lipschitz bounded. F can be any arbitary function and can output any value. In JS, D, discriminator is a binary classifier whose output is 0 1.

Question: How close are the two Objectives ?
Let $D(x) = \sigma(F(x))$, where F(x) is the logits of Generator G.

- JS

$$JS(p_g||p_d) = \max_D \mathop{\mathbb{E}}_{x\sim p_{data}}[\underbrace{-log(1 + e^{-F(x)})}_{\text{Soft negative part}}] + \mathop{\mathbb{E}}_{x^{'}\sim p_g}[\underbrace{-log(1 + e^{F(x^{'})})}_{\text{Soft positive part}}]$$

$$= \max_{F} \mathop{\mathbb{E}}_{x \sim p_{data}} [\lfloor F(x) \rfloor_-] - \mathop{\mathbb{E}}_{x' \sim p_g} [\lfloor F(x') \rfloor_+]$$

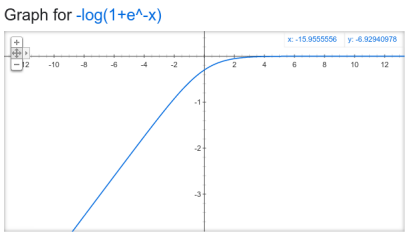


Figure 6: Soft negative part

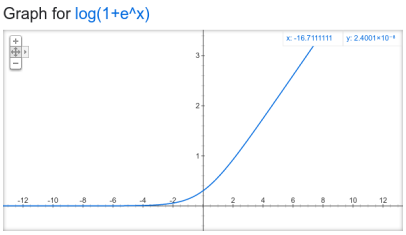


Figure 7: Soft positive part

We see that the soft negative and soft positive part are soft versions of laterally inverted inverse ReLU and normal ReLU respectively.

- Wasserstein

$$W(p_g, p_d) = \max_{||F||_{L \leq 1}} \mathop{\mathbb{E}}_{x \sim p_{data}} [F(x)] - \mathop{\mathbb{E}}_{x' \sim p_g} [F(x')]$$

Wasserstein loss does not have the soft postive and soft negative part as in JS divergence. JS gradient saturates to zero when F(x) is negative for data and F(x') is positive for generated distribution. Wasserstein does not suffer from the vanishing gradient problem we observe in JS divergence. WGAN tries to maximize the difference between the output for real data and generated data. As discriminator can output arbitary values it's output cannot be used for classifying between real and fake samples as in JS divergence by setting a threshold.
Lipschitz constraint in Wasserstein is a novel technique which prevents discriminator from becoming arbitarily very good. If F gets very good we see vanishing gradients as in JS.

How do we ensure that the discriminator satisfies 1-Lipschitz constraint. We will see in the next section that we rely on approximation techniques.

4.1 Clipping

The original WGAN paper presented the clipping as an approximation technique. Clipping the weights of the discriminator is an idea that has been used a lot in the deep learning community as a way to prevent things to get too large.

The theoretical insight for this practice is that if we have a neural network with bounded weights, then the function will be Lipschitz. The slope of the linear pieces of the neural network function are limited, thus the variation of the combination (made with ReLU activation or something else) of those pieces cannot be harder than the one determined by the bounded weights.

We clip weight and not gradient because gradient clipping has a different motivation, it is used to stabilize the training. Let's say that the gradient is clipped to 1. If we do not clip the weight, we could go to infinity in the direction pointed by the gradient, making the slope increase very fast and the learning diverge.

Instead of clipping, they could have used $L2$ norm on the weights in order to regularize them. The only downfall is that applying $L2$ regularization to the weights is changing the objective by shifting the value of the equilibrium. Clipping does not have that shifting effect, it is a $L\infty$ norm constraint.

However, with this method, it is hard to exactly control the Lipschitzness of the function. We are constraining, but with the wrong bowl. As an example, if we have $f(x) = \theta_L \cdot ... \cdot \theta_1 x$ with $|\theta_l| < c$ a linear neural network, then the function f is c^L-Lipschitz. In other words, we make the Lipschitz constraints grow (or vanish) exponentially with the depth.

Algorithm 1 WGAN, our proposed algorithm. All experiments in the paper used the default values $\alpha = 0.00005$, $c = 0.01$, $m = 64$, $n_{\text{critic}} = 5$.

Require: : α, the learning rate. c, the clipping parameter. m, the batch size. n_{critic}, the number of iterations of the critic per generator iteration.
Require: : w_0, initial critic parameters. θ_0, initial generator's parameters.
1: **while** θ has not converged **do**
2: **for** $t = 0, ..., n_{\text{critic}}$ **do**
3: Sample $\{x^{(i)}\}_{i=1}^{m} \sim \mathbb{P}_r$ a batch from the real data.
4: Sample $\{z^{(i)}\}_{i=1}^{m} \sim p(z)$ a batch of prior samples.
5: $g_w \leftarrow \nabla_w \left[\frac{1}{m}\sum_{i=1}^{m} f_w(x^{(i)}) - \frac{1}{m}\sum_{i=1}^{m} f_w(g_\theta(z^{(i)}))\right]$
6: $w \leftarrow w + \alpha \cdot \text{RMSProp}(w, g_w)$
7: $w \leftarrow \text{clip}(w, -c, c)$
8: **end for**
9: Sample $\{z^{(i)}\}_{i=1}^{m} \sim p(z)$ a batch of prior samples.
10: $g_\theta \leftarrow -\nabla_\theta \frac{1}{m}\sum_{i=1}^{m} f_w(g_\theta(z^{(i)}))$
11: $\theta \leftarrow \theta - \alpha \cdot \text{RMSProp}(\theta, g_\theta)$
12: **end while**

Figure 8: WGAN algorithm with weight clipping is simple to implement (Arjovsky et al. [1])

In the original paper, they use this method because it is fast and simple to implement as the algorithm is basically the same, with a line added to constraint the weights. This line correspond to the number 7 in Figure 8. How can we choose a good clipping bound c? They picked theirs, $c = 0.01$, without extensive justification. They are well aware that weight clipping is a terrible way to enforce Lipschitz constraints. Having a clipping parameter that is too large can slow down the learning because the weights will take a long time to get to their limit. On the other side, having a clipping parameter that is too small can cause vanishing gradient issues.

4.2 Gradient Penalty

The idea behind gradient penalty is that if we have a differenciable function, bounding the norm of the gradient is equivalent to having a Lipschitz function.

This method is easy to implement, it consists in adding a term to the loss to regularize the norm of the gradient of D.

$$\tilde{\mathcal{L}}_D = \mathcal{L}_D + \lambda \mathbb{E}_{\tilde{x} \sim \epsilon P_d + (1-\epsilon) p_g} \left[\left(\|\nabla_x D(\tilde{x})\|_2 - 1 \right)^2 \right] \tag{6}$$

We can see in the equation 6 that the loss is minimized when the norm of the gradient with respect to the input is close to one. This technique is tractable, but it has some disadvantages, such as the fact that it is implicitly controlled. We have to tune the λ parameter, and it is not following the true constraint because we are enforcing the gradient to be close to one everywhere. The true constraint would be for the gradient to be at most one instead of one everywhere. Also, if the λ parameter is too large and makes the $\mathcal{L}_D$ term useless, it creates bad attracting points and decreases the performance.

To backpropagate this quantity, there is a trick.

$$\nabla_\varphi \left(\|\nabla_x D_\varphi(x)\|^2 \right) = 2 < \nabla_x D_\varphi(x), \nabla_x \nabla_\varphi D_\varphi(x) >$$

If we do not use the trick, we might get stuck because the second term in the inner product is a Jacobian of the size of the input space times the number of parameters. The trick consists in saying that the backpropagation for $f_u(\varphi) =<$

$\nabla_x D_\varphi(x), u >$, where u is fixed to $\nabla_x D_\varphi(x)$, a quantity that we computed already by a first backpropagation, is

$$\begin{aligned}\nabla_\varphi f_u(\varphi) &= \nabla_\varphi(< \nabla_x D_\varphi(x), u >) \\ &=< \nabla_\varphi \nabla_x D_\varphi(x), \nabla_x D_\varphi(x) > \\ &= \nabla_\varphi \left(\|\nabla_x D_\varphi(x)\|^2 \right)\end{aligned}$$

So, by backpropagating twice, we get the result without having to explicitly compute a Hessian matrix.

GG: the original idea of gradient penalty and the practical implementation by double backprop has been first introduced by by this paper: http://yann.lecun.com/exdb/publis/pdf/drucker-lecun-92.pdf
Note: This trick allows you to compute efficiently

$$< \nabla_\varphi \nabla_x D_\varphi(x), u > \tag{7}$$

for any vector u.

Gauthier's take on the gradient penalty There are some unusual facts about gradient penalty. For example, in optimization, we usually regularize with the square of the norm, because it is smooth and the norm alone is not smooth. Also, this technique penalizes gradients smaller than one, but they might occur in Lipschitz functions. i.e. we may want a function that is flat in some areas.

An other remark: there is some other papers that propose to penalyze the squared norm of the gradient (with respect to the parameters this time) [2]

$$\nabla_\theta \|L(\theta, \phi)\|^2 \quad \text{where} \quad L(\theta, \phi) = \frac{1}{2} \|\nabla \left(\mathbb{E}_{x \sim p_d}[\log D(x)] + \mathbb{E}_{z \sim p_z}[\log(1 - D(G(z))] \right) \|_2^2 \tag{8}$$

it is non trivial to get unbiased estimator of this quantity. In the numerics of GANs paper [2] they use a biased quantity. If $1/|B| \sum \nabla \mathcal{L}(x_i)$ is an unbiased stochastic estimate of the gradient, with B the minibatch, $\|1/|B| \sum \nabla \mathcal{L}(x_i, \theta)\|^2$ is not an unbiased estimate of the norm of the gradient. A simple way to see that is because we are looking at the norm of the minibatches. Taken altogether, the estimate of the gradient could, let's say, be 0, but taken individually, the estimate over B of $\|1/|B| \sum \nabla \mathcal{L}(x_i, \theta)\|^2$ could send us in different directions. The core of this remark is the inequality $\mathbb{E}[X^2] \neq \mathbb{E}[X]^2$.
Overall, a potential alternative for gradient penalty could be the following.

$$\mathbb{E}_{\tilde{x} \sim \epsilon P_d + (1-\epsilon) p_g} \left[\|\nabla_x D(\tilde{x})\|_2^2 \right] \tag{9}$$

This would penalize the big values of the gradient, thus allowing the smaller values that naturally occur.

4.3 Spectral Normalization

The idea here is really simple. We basically want to compute the Lipschitz constant of a given network and then re-normalize everything by this constant.

It is hard to compute the Lipschitz constant, so we find an upper bound for it instead.

$$\|\sigma(W_L \cdots \sigma(W_1 x))\|_{Lip} \leq \|W_L\| \cdots \|W_1\| \tag{10}$$

We have the σ that are 1-Lipschitz non-linearities, and the upper bound is composed of the product of the spectral matrix norms $\|W_l\|$. Finding the spectral matrices is non trivial, but we have efficient algorithms to estimate it.

Since this technique is a better control of the Lipschitz, it gives better results in practice. It is difficult to implement, but they did it for us (and it is now available in PyTorch), so a lot of people are citing this paper [3]. It is still an approximation of the upper bound and it is a little slower to optimize, but it overall works very well.

Now, let us see how we compute the spectral matrix norm.

$$W = U^T DV;\ U^T U = I;\ V^T V = I;\ D = \begin{pmatrix} \sigma_1 & 0 & \dots & 0 & 0 \\ 0 & \sigma_2 & \dots & 0 & 0 \\ 0 & 0 & \ddots & 0 & \\ 0 & 0 & \dots & \sigma_{n-1} & 0 \\ 0 & 0 & \dots & 0 & \sigma_n \end{pmatrix} \mathbf{0}$$

In the matrix D, the elements are real numbers. If W is a square matrix, then it is a diagonal matrix. Otherwise, there are columns of zeros.

We can use one of those two definitions to find $\|W\|$.

$$\|W\| = \max_i(\sigma_i)$$
$$\|W\| = \max_{\|u\|=1} \|Wu\| = \|Wu^*\|$$

It is either the maximum singular value, that is sort of the Lipschitz constant of the matrix W, or the the maximal inflation a unit vector can take in a direction.

How to compute $\|W\|$ efficiently? A first solution could be to compute SVD and output $\max_i(\sigma_i)$, but the issue is that it has a complexity of d^2 in the case where W is a $d \times d$ matrix. To have a better complexity of $\mathcal{O}(d)$, there is a trick : to find a approximate value by using an iterative algorithm.

$$u_0 \sim \mathcal{N}(0, I)$$
$$u_{t+1} = \frac{Wu_t}{\|Wu_t\|}$$

This iterative algorithm will lead us to $\|Wu^*\|$ by choosing the direction of the biggest inflation through the consecutive normalizations (u_t converges towards u^* very quickly). This is known as the "Power Method".

References

[1] M. Arjovsky, S. Chintala, and L. Bottou. Wasserstein gan. 2017.

[2] L. Mescheder, S. Nowozin, and A. Geiger. The numerics of gans. *arXiv preprint arXiv:1705.10461*, 2017.

[3] T. Miyato, T. Kataoka, M. Koyama, and Y. Yoshida. Spectral normalization for generative adversarial networks, 2018.

[4] G. Peyré and M. Cuturi. Computational optimal transport. URL https://optimaltransport.github.io/.

[5] R. J. Vanderbei. Linear programming: Foundations and extensions, 2001.

[6] C. Villani. *Optimal Transport: Old and New*. Springer Science Business Media, 2008.

A Regularity assumption[3]

Assumption 1. *Let $g : \mathcal{Z} \times \mathbb{R}^d \to \mathcal{X}$ be locally Lipschitz between finite dimensional vector spaces. We will denote $g_\theta(z)$ it's evaluation on coordinates (z, θ). We say that g satisfies assumption 1 for a certain probability distribution p over $\mathcal{Z}$ if there are local Lipschitz constants $L(\theta, z)$ such that*

$$\mathbb{E}_{z\sim p}[L(\theta, z)] < +\infty$$

B Example: Failure mode of Wasserstein measure in high dimension

$y = x + \epsilon Noise$
$W(x, y) = ||x - y|| = \epsilon d$ This can be large in high dimension when it should have been small