

Wasserstein Generative Adversarial Networks

What does it mean to *learn* a probability distribution?

Stefano Gualandi

Università di Pavia, Dipartimento di Matematica

email: stefano.gualandi@unipv.it
twitter: [@famo2spaghi](https://twitter.com/famo2spaghi)
blog: <http://stegua.github.com>
web: <http://matematica.unipv.it/gualandi/>

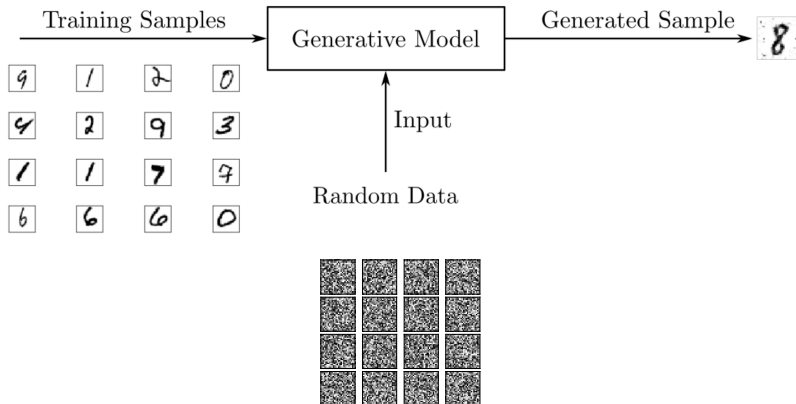
Outline

1 Introduction

2 Wasserstein GAN

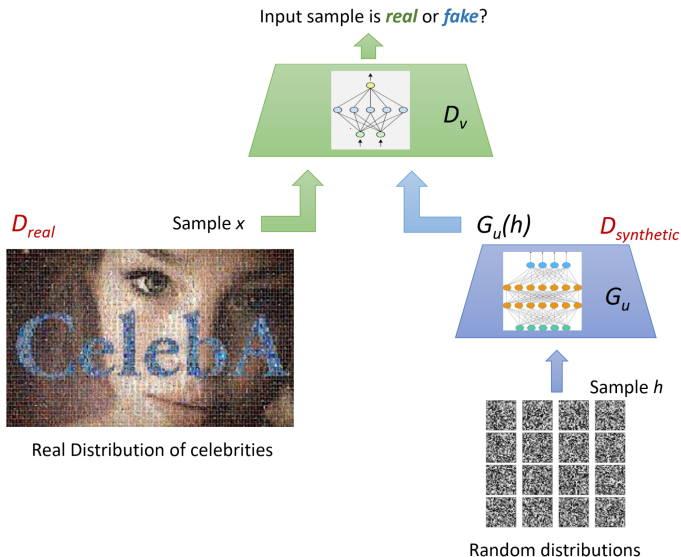
Using Deep NNs as Generative Models

This is an example of **Unsupervised Learning**



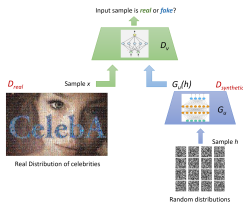
Picture from: <https://realpython.com/generative-adversarial-networks/>

The GAN game



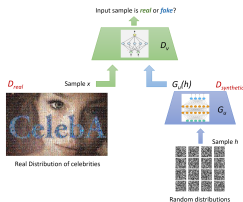
Generators and Discriminators

- $\{G_u \mid u \in \mathcal{U} \subset \mathbb{R}^p\}$, where $G_u : \mathbb{R}^\ell \rightarrow \mathbb{R}^d$, is the class of generators with parameters u . $\mathcal{U}$ is a subset of the unit ball.

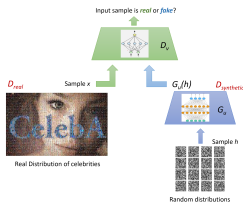


Generators and Discriminators

- $\{G_u \mid u \in \mathcal{U} \subset \mathbb{R}^p\}$, where $G_u : \mathbb{R}^\ell \rightarrow \mathbb{R}^d$, is the **class of generators with parameters u** . $\mathcal{U}$ is a subset of the unit ball.
- The generator G_u defines a distribution $\mathcal{D}_{G_u}$: generate h and then apply G_u to get $x = G_u(h)$ of $\mathcal{D}_{G_u}$.

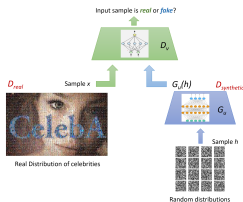


Generators and Discriminators



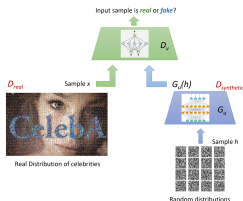
- $\{G_u \mid u \in \mathcal{U} \subset \mathbb{R}^p\}$, where $G_u : \mathbb{R}^\ell \rightarrow \mathbb{R}^d$, is the **class of generators with parameters u** . $\mathcal{U}$ is a subset of the unit ball.
- The generator G_u defines a distribution $\mathcal{D}_{G_u}$: generate h and then apply G_u to get $x = G_u(h)$ of $\mathcal{D}_{G_u}$.
- $\{D_v \mid v \in \mathcal{V}\}$, where $D_v : \mathbb{R}^d \rightarrow [0..1]$, is the **class of discriminators with parameters v** .

Generators and Discriminators



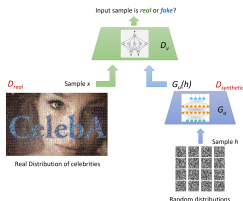
- $\{G_u \mid u \in \mathcal{U} \subset \mathbb{R}^p\}$, where $G_u : \mathbb{R}^\ell \rightarrow \mathbb{R}^d$, is the **class of generators with parameters u** . $\mathcal{U}$ is a subset of the unit ball.
- The generator G_u defines a distribution $\mathcal{D}_{G_u}$: generate h and then apply G_u to get $x = G_u(h)$ of $\mathcal{D}_{G_u}$.
- $\{D_v \mid v \in \mathcal{V}\}$, where $D_v : \mathbb{R}^d \rightarrow [0..1]$, is the **class of discriminators with parameters v** .
- $D_v(x)$ is the probability that x comes from $\mathcal{D}_{real}$.

Generators and Discriminators



- $\{G_u \mid u \in \mathcal{U} \subset \mathbb{R}^p\}$, where $G_u : \mathbb{R}^\ell \rightarrow \mathbb{R}^d$, is the **class of generators with parameters u** . $\mathcal{U}$ is a subset of the unit ball.
- The generator G_u defines a distribution $\mathcal{D}_{G_u}$: generate h and then apply G_u to get $x = G_u(h)$ of $\mathcal{D}_{G_u}$.
- $\{D_v \mid v \in \mathcal{V}\}$, where $D_v : \mathbb{R}^d \rightarrow [0..1]$, is the **class of discriminators with parameters v** .
- $D_v(x)$ is the probability that x comes from $\mathcal{D}_{\text{real}}$.
- $1 - D_v(x)$ is the probability that x comes from $\mathcal{D}_{G_u}$.

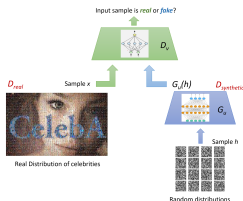
Generators and Discriminators



- $\{G_u \mid u \in \mathcal{U} \subset \mathbb{R}^p\}$, where $G_u : \mathbb{R}^\ell \rightarrow \mathbb{R}^d$, is the **class of generators with parameters u** . $\mathcal{U}$ is a subset of the unit ball.
- The generator G_u defines a distribution $\mathcal{D}_{G_u}$: generate h and then apply G_u to get $x = G_u(h)$ of $\mathcal{D}_{G_u}$.
- $\{D_v \mid v \in \mathcal{V}\}$, where $D_v : \mathbb{R}^d \rightarrow [0..1]$, is the **class of discriminators with parameters v** .
- $D_v(x)$ is the probability that x comes from $\mathcal{D}_{\text{real}}$.
- $1 - D_v(x)$ is the probability that x comes from $\mathcal{D}_{G_u}$.
- We assume G_u and D_v are L -Lipschitz with respect to the parameters. For all $u, u' \in \mathcal{U}$ and any input h :

$$\|G_u(h) - G_{u'}(h)\| \leq L \|u - u'\|.$$

Generators and Discriminators



- $\{G_u \mid u \in \mathcal{U} \subset \mathbb{R}^p\}$, where $G_u : \mathbb{R}^\ell \rightarrow \mathbb{R}^d$, is the **class of generators with parameters u** . $\mathcal{U}$ is a subset of the unit ball.
- The generator G_u defines a distribution $\mathcal{D}_{G_u}$: generate h and then apply G_u to get $x = G_u(h)$ of $\mathcal{D}_{G_u}$.
- $\{D_v \mid v \in \mathcal{V}\}$, where $D_v : \mathbb{R}^d \rightarrow [0..1]$, is the **class of discriminators with parameters v** .
- $D_v(x)$ is the probability that x comes from $\mathcal{D}_{real}$.
- $1 - D_v(x)$ is the probability that x comes from $\mathcal{D}_{G_u}$.
- We assume G_u and D_v are L -Lipschitz with respect to the parameters. For all $u, u' \in \mathcal{U}$ and any input h :

$$\|G_u(h) - G_{u'}(h)\| \leq L \|u - u'\|.$$

Training a GAN: a min-max game

The training of a GAN consists in optimizing the objective function:

$$\min_{u \in \mathcal{U}} \max_{v \in \mathcal{V}} \mathbb{E}_{x \sim \mathcal{D}_{real}} [\log D_v(x)] + \mathbb{E}_{x \sim \mathcal{D}_{G_u}} [\log(1 - D_v(x))] \quad (1)$$

Objective function

Training a GAN: a min-max game

The training of a GAN consists in optimizing the objective function:

$$\min_{u \in \mathcal{U}} \max_{v \in \mathcal{V}} \mathbb{E}_{x \sim \mathcal{D}_{real}} [\log D_v(x)] + \mathbb{E}_{x \sim \mathcal{D}_{G_u}} [\log(1 - D_v(x))] \quad (2)$$

Notice that $\log x \rightarrow -\infty$ as $x \rightarrow 0$ (numerical issues).

Objective function

Training a GAN: a min-max game

The training of a GAN consists in optimizing the objective function:

$$\min_{u \in \mathcal{U}} \max_{v \in \mathcal{V}} \mathbb{E}_{x \sim \mathcal{D}_{real}} [\log D_v(x)] + \mathbb{E}_{x \sim \mathcal{D}_{G_u}} [\log(1 - D_v(x))] \quad (2)$$

Notice that $\log x \rightarrow -\infty$ as $x \rightarrow 0$ (numerical issues).

We can replace $\log(\cdot)$ with a monotone function $\phi : [0..1] \rightarrow \mathbb{R}$:

$$\min_{u \in \mathcal{U}} \max_{v \in \mathcal{V}} \mathbb{E}_{x \sim \mathcal{D}_{real}} [\phi(D_v(x))] + \mathbb{E}_{x \sim \mathcal{D}_{G_u}} [\phi(1 - D_v(x))] \quad (3)$$

We call $\phi(\cdot)$ the *measuring function*.

Objective function

Training a GAN: a min-max game

The training of a GAN consists in optimizing the objective function:

$$\min_{u \in \mathcal{U}} \max_{v \in \mathcal{V}} \mathbb{E}_{x \sim \mathcal{D}_{real}} [\log D_v(x)] + \mathbb{E}_{x \sim \mathcal{D}_{G_u}} [\log(1 - D_v(x))] \quad (2)$$

Notice that $\log x \rightarrow -\infty$ as $x \rightarrow 0$ (numerical issues).

We can replace $\log(\cdot)$ with a monotone function $\phi : [0..1] \rightarrow \mathbb{R}$:

$$\min_{u \in \mathcal{U}} \max_{v \in \mathcal{V}} \mathbb{E}_{x \sim \mathcal{D}_{real}} [\phi(D_v(x))] + \mathbb{E}_{x \sim \mathcal{D}_{G_u}} [\phi(1 - D_v(x))] \quad (3)$$

We call $\phi(\cdot)$ the *measuring function*.

In practice, we have m training samples $x_1, \dots, x_m \sim \mathcal{D}_{real}$, and instead of the expectation, we use the *empirical distribution*:

$$\frac{1}{m} \sum_{i=1}^m \phi(D_v(x_i))$$

Training Algorithm (Goodfellow et al 2014)

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k , is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

for number of training iterations **do**

for k steps **do**

- Sample minibatch of m noise samples $\{z^{(1)}, \dots, z^{(m)}\}$ from noise prior $p_g(z)$.
- Sample minibatch of m examples $\{x^{(1)}, \dots, x^{(m)}\}$ from data generating distribution $p_{\text{data}}(x)$.
- Update the discriminator by ascending its stochastic gradient:

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

end for

- Sample minibatch of m noise samples $\{z^{(1)}, \dots, z^{(m)}\}$ from noise prior $p_g(z)$.
- Update the generator by descending its stochastic gradient:

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

end for

The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

GAN Advantages

- 1 **Sharp Images:** GANs produce sharper images than other generative models.

GAN Advantages

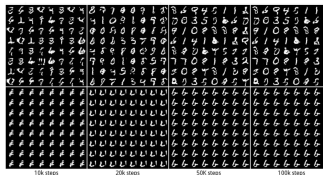
- 1 **Sharp Images:** GANs produce sharper images than other generative models.
- 2 **Configurable size:** The latent random variable size is unrestricted, enriching the generator search space.

GAN Advantages

- 1 Sharp Images:** GANs produce sharper images than other generative models.
- 2 Configurable size:** The latent random variable size is unrestricted, enriching the generator search space.
- 3 Versatile generator:** The GAN framework can support many different generator networks (*it's a GAN zoo nowadays...*)

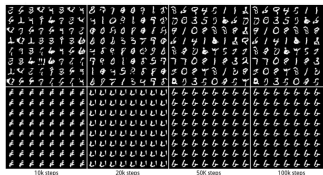
GAN Issues

- 1 Mode collapse:** The generator tends to produce a specific pattern which fools the discriminator.



GAN Issues

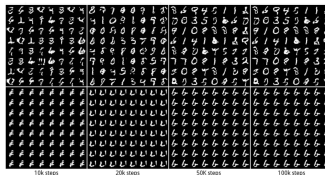
- 1 Mode collapse:** The generator tends to produce a specific pattern which fools the discriminator.



- 2 Vanishing gradients:** The training step of the generator backpropagates very low gradients (no info!).

GAN Issues

- Mode collapse:** The generator tends to produce a specific pattern which fools the discriminator.



- Vanishing gradients:** The training step of the generator backpropagates very low gradients (no info!).
- Instability:** The parameters u and v oscillate during training and they do not converge toward an equilibrium (low quality images in output).

Main Motivation

Unsupervised learning

QUESTION: What does it mean to learn a probability distribution?

Main Motivation

Unsupervised learning

QUESTION: What does it mean to learn a probability distribution?

CLASSICAL ANSWER: To learn a probability density.

Main Motivation

Unsupervised learning

QUESTION: What does it mean to learn a probability distribution?

CLASSICAL ANSWER: To learn a probability density.

Learning densities

The standard approach defines a parametric family of densities $(P_\theta)_{\theta \in \mathbb{R}^d}$, and if we have real data samples $\{x^{(i)}\}_{i=1}^m$, we solve the **maximum likelihood problem**:

$$\max_{\theta \in \mathbb{R}^d} \frac{1}{m} \sum_{i=1}^m \log P_\theta(x^{(i)})$$

If the real data distribution $\mathbb{P}_r$ admits a density and $\mathbb{P}_\theta$ is the distribution of P_θ , then this amounts to minimize the Kullback-Leibler divergence:

$$KL(\mathbb{P}_r || \mathbb{P}_\theta) := \int \log \left(\frac{P_r(x)}{P_\theta(x)} \right) P_r(x) d\mu(x)$$

Main Motivation

Learning Sample Generators

Rather than estimating $\mathbb{P}_r$, we can define a random variable Z with a fixed distribution $p(z)$ and pass it in input to a parametric function $g_\theta : \mathcal{Z} \rightarrow \mathcal{X}$ (e.g. a Neural Network) that directly generate samples with a distribution $\mathbb{P}_\theta$.

Main Motivation

Learning Sample Generators

Rather than estimating $\mathbb{P}_r$, we can define a random variable Z with a fixed distribution $p(z)$ and pass it in input to a parametric function $g_\theta : \mathcal{Z} \rightarrow \mathcal{X}$ (e.g. a Neural Network) that directly generate samples with a distribution $\mathbb{P}_\theta$.

By changing θ we can change $\mathbb{P}_\theta$ to make it as close as possible to $\mathbb{P}_r$.

Main Motivation

Learning Sample Generators

Rather than estimating $\mathbb{P}_r$, we can define a random variable Z with a fixed distribution $p(z)$ and pass it in input to a parametric function $g_\theta : \mathcal{Z} \rightarrow \mathcal{X}$ (e.g. a Neural Network) that **directly generate samples with a distribution $\mathbb{P}_\theta$** .

By changing θ we can change $\mathbb{P}_\theta$ to make it as close as possible to $\mathbb{P}_r$.

REMARK: The ability to generate samples is often more useful than knowing the numerical value of the density.

Distances and Divergences

To measure how similar the model distribution and the real distributions are we need to define a **distance** (or *divergence*) $\rho(\mathbb{P}_\theta, \mathbb{P}_r)$.

Let $\mathcal{X}$ be a compact metric set such as the space of images $[0, 1]^d$. Examples:

- **Total Variation (TV) distance:**

$$\delta(\mathbb{P}_r, \mathbb{P}_g) = \sup_{A \in \Sigma} |\mathbb{P}_r(A) - \mathbb{P}_g(A)|$$

Distances and Divergences

To measure how similar the model distribution and the real distributions are we need to define a **distance** (or *divergence*) $\rho(\mathbb{P}_\theta, \mathbb{P}_r)$.

Let $\mathcal{X}$ be a compact metric set such as the space of images $[0, 1]^d$. Examples:

- **Total Variation (TV) distance:**

$$\delta(\mathbb{P}_r, \mathbb{P}_g) = \sup_{A \in \Sigma} |\mathbb{P}_r(A) - \mathbb{P}_g(A)|$$

- **Kullback-Leibler (KL) divergence:**

$$KL(\mathbb{P}_r || \mathbb{P}_g) := \int \log \left(\frac{P_r(x)}{P_g(x)} \right) P_r(x) d\mu(x)$$

Distances and Divergences

To measure how similar the model distribution and the real distributions are we need to define a **distance** (or *divergence*) $\rho(\mathbb{P}_\theta, \mathbb{P}_r)$.

Let $\mathcal{X}$ be a compact metric set such as the space of images $[0, 1]^d$. Examples:

- **Total Variation (TV) distance:**

$$\delta(\mathbb{P}_r, \mathbb{P}_g) = \sup_{A \in \Sigma} |\mathbb{P}_r(A) - \mathbb{P}_g(A)|$$

- **Kullback-Leibler (KL) divergence:**

$$KL(\mathbb{P}_r || \mathbb{P}_g) := \int \log \left(\frac{P_r(x)}{P_g(x)} \right) P_r(x) d\mu(x)$$

- **Jensen-Shannon (JS) divergence:**

$$JS(\mathbb{P}_r, \mathbb{P}_g) := KL(\mathbb{P}_r || \mathbb{P}_m) + KL(\mathbb{P}_g || \mathbb{P}_m)$$

where $\mathbb{P}_m$ is the mixture $\frac{\mathbb{P}_r + \mathbb{P}_g}{2}$.

Distances and Divergences

To measure how similar the model distribution and the real distributions are we need to define a **distance** (or *divergence*) $\rho(\mathbb{P}_\theta, \mathbb{P}_r)$.

Let $\mathcal{X}$ be a compact metric set such as the space of images $[0, 1]^d$. Examples:

- **Earth-Mover (EM)** distance or **Wasserstein-1**:

$$W(\mathbb{P}_r, \mathbb{P}_g) = \inf_{\gamma \in \Pi(\mathbb{P}_r, \mathbb{P}_g)} \mathbb{E}_{(x,y) \sim \gamma} [\|x - y\|]$$

where $\Pi(\mathbb{P}_r, \mathbb{P}_g)$ denotes the set of all joint distributions $\gamma(x, y)$ whose marginals are respectively $\mathbb{P}_r$ and $\mathbb{P}_g$.

Convergence results

Example 1 (Learning parallel lines). Let $Z \sim U[0, 1]$ the uniform distribution on the unit interval. Let $\mathbb{P}_0$ be the distribution of $(0, Z) \in \mathbb{R}^2$ (a 0 on the x-axis and the random variable Z on the y-axis), uniform on a straight vertical line passing through the origin. Now let $g_\theta(z) = (\theta, z)$ with θ a single real parameter. It is easy to see that in this case,

- $W(\mathbb{P}_0, \mathbb{P}_\theta) = |\theta|,$
- $JS(\mathbb{P}_0, \mathbb{P}_\theta) = \begin{cases} \log 2 & \text{if } \theta \neq 0, \\ 0 & \text{if } \theta = 0, \end{cases}$
- $KL(\mathbb{P}_\theta \| \mathbb{P}_0) = KL(\mathbb{P}_0 \| \mathbb{P}_\theta) = \begin{cases} +\infty & \text{if } \theta \neq 0, \\ 0 & \text{if } \theta = 0, \end{cases}$
- and $\delta(\mathbb{P}_0, \mathbb{P}_\theta) = \begin{cases} 1 & \text{if } \theta \neq 0, \\ 0 & \text{if } \theta = 0. \end{cases}$

When $\theta_t \rightarrow 0$, the sequence $(\mathbb{P}_{\theta_t})_{t \in \mathbb{N}}$ converges to $\mathbb{P}_0$ under the EM distance, but does not converge at all under either the JS, KL, reverse KL, or TV divergences.

Convergence results

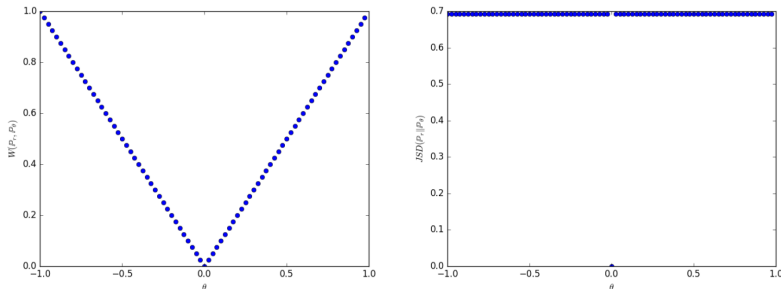


Figure 1: These plots show $\rho(\mathbb{P}_\theta, \mathbb{P}_0)$ as a function of θ when ρ is the EM distance (left plot) or the JS divergence (right plot). The EM plot is continuous and provides a usable gradient everywhere. The JS plot is not continuous and does not provide a usable gradient.

Convergence results

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 $g : \mathcal{Z} \times \mathbb{R}^d \rightarrow \mathcal{X}$ 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 1, 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.*

Convergence results

Theorem 3. *Let $\mathbb{P}_r$ be any distribution. Let $\mathbb{P}_\theta$ be the distribution of $g_\theta(Z)$ with Z a random variable with density p and g_θ a function satisfying assumption 1. Then, there is a solution $f : \mathcal{X} \rightarrow \mathbb{R}$ to the problem*

$$\max_{\|f\|_L \leq 1} \mathbb{E}_{x \sim \mathbb{P}_r}[f(x)] - \mathbb{E}_{x \sim \mathbb{P}_\theta}[f(x)]$$

and we have

$$\nabla_\theta W(\mathbb{P}_r, \mathbb{P}_\theta) = -\mathbb{E}_{z \sim p(z)}[\nabla_\theta f(g_\theta(z))]$$

when both terms are well-defined.

Algorithm: Original W-GAN with Clipping

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  $\theta$  has not converged do
2:   for  $t = 0, \dots, n_{\text{critic}}$  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:      $w \leftarrow w + \alpha \cdot \text{RMSPProp}(w, g_w)$ 
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:   $\theta \leftarrow \theta - \alpha \cdot \text{RMSPProp}(\theta, g_\theta)$ 
12: end while
```

Algorithm: Original W-GAN with Gradient Penalty

Algorithm 1 WGAN with gradient penalty. We use default values of $\lambda = 10$, $n_{\text{critic}} = 5$, $\alpha = 0.0001$, $\beta_1 = 0$, $\beta_2 = 0.9$.

Require: The gradient penalty coefficient λ , the number of critic iterations per generator iteration n_{critic} , the batch size m , Adam hyperparameters α, β_1, β_2 .

Require: initial critic parameters w_0 , initial generator parameters θ_0 .

```

1: while  $\theta$  has not converged do
2:   for  $t = 1, \dots, n_{\text{critic}}$  do
3:     for  $i = 1, \dots, m$  do
4:       Sample real data  $\mathbf{x} \sim \mathbb{P}_r$ , latent variable  $\mathbf{z} \sim p(\mathbf{z})$ , a random number  $\epsilon \sim U[0, 1]$ .
5:        $\hat{\mathbf{x}} \leftarrow G_{\theta}(\mathbf{z})$ 
6:        $\tilde{\mathbf{x}} \leftarrow \epsilon \mathbf{x} + (1 - \epsilon) \hat{\mathbf{x}}$ 
7:        $L^{(i)} \leftarrow D_w(\hat{\mathbf{x}}) - D_w(\mathbf{x}) + \lambda(\|\nabla_{\hat{\mathbf{x}}} D_w(\hat{\mathbf{x}})\|_2 - 1)^2$ 
8:     end for
9:      $w \leftarrow \text{Adam}(\nabla_w \frac{1}{m} \sum_{i=1}^m L^{(i)}, w, \alpha, \beta_1, \beta_2)$ 
10:  end for
11:  Sample a batch of latent variables  $\{\mathbf{z}^{(i)}\}_{i=1}^m \sim p(\mathbf{z})$ .
12:   $\theta \leftarrow \text{Adam}(\nabla_{\theta} \frac{1}{m} \sum_{i=1}^m -D_w(G_{\theta}(\mathbf{z}^{(i)})), \theta, \alpha, \beta_1, \beta_2)$ 
13: end while
```

Google Colab 3: Wasserstein GAN with Clipping

Google Colab 4: W-GAN with Gradient Penalty

TensorBoard IMAGES

☐ Show actual image size

Brightness adjustment
RESET

Contrast adjustment
RESET

Runs

Write a regex to filter runs

☒ real ☐ real
☒ fake ☐ fake

TOGGLE ALL RUNS

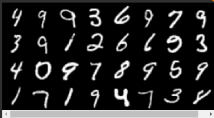
logs

tag: fake
step 26
Tue May 03 2022 17:53:10 GMT+0200 (Ora legale dell'Europa centrale)



Real

Real
tag: Real
step 464
Tue May 03 2022 18:01:56 GMT+0200 (Ora legale dell'Europa centrale)



Google Colab 4: W-GAN with Gradient Penalty

TensorBoard IMAGES

☐ Show actual image size

Brightness adjustment
RESET

Contrast adjustment
RESET

Runs
Write a regex to filter runs

☒ real
☒ fake

TOGGLE ALL RUNS

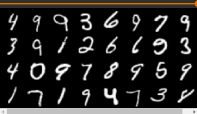
logs

tag: fake
step: 100
Tue May 03 2022 17:54:40 GMT+0200 (Ora legale dell'Europa centrale)



Real

Real
tag: Real
step: 464
Tue May 03 2022 18:01:56 GMT+0200 (Ora legale dell'Europa centrale)



Google Colab 4: W-GAN with Gradient Penalty

TensorBoard IMAGES

☐ Show actual image size

Brightness adjustment
RESET

Contrast adjustment
RESET

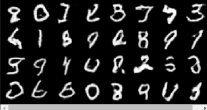
Runs
Write a regex to filter runs

☒ real
☒ fake

TOGGLE ALL RUNS

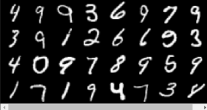
logs

tag: Fake
step 464
Tue May 03 2022 18:01:56 GMT+0200 (Ora legale dell'Europa centrale)

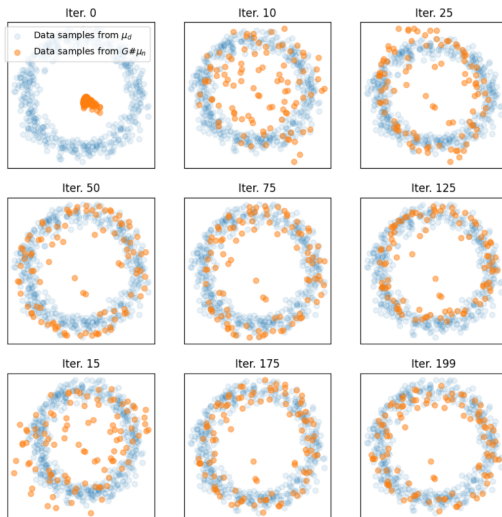


Real

Real
tag: Real
step 464
Tue May 03 2022 18:01:56 GMT+0200 (Ora legale dell'Europa centrale)



Google Colab 5: Minibatch W-GAN by Network Simplex



Main References

- [Wasserstein GAN](https://arxiv.org/pdf/1701.07875.pdf). By Martin Arjovsky, Soumith Chintala, and Leon Bottou.
<https://arxiv.org/pdf/1701.07875.pdf>
- [Improved Training of Wasserstein GANs](https://arxiv.org/pdf/1704.00028.pdf). By Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, Aaron Courville
<https://arxiv.org/pdf/1704.00028.pdf>
- [Learning with minibatch Wasserstein: asymptotic and gradient properties](http://proceedings.mlr.press/v108/fatras20a/fatras20a.pdf). By Fatras, K., Zine, Y., Flamary, R., Gribonval, R., Courty, N.
<http://proceedings.mlr.press/v108/fatras20a/fatras20a.pdf>