Week 8: Generative Adversarial Networks

Instructor: Ruixuan Wang
wangruix5@mail.sysu.edu.cn

School of Data and Computer Science
Sun Yat-Sen University

18 April, 2019
1. Generative adversarial network

2. Wasserstein GANs
Generative model

- Objective: generate new data by learning from training data
- Various applications: realistic new designs, super-resolution, colorization, etc.

Data augmentation: more realistic data for model training
Data mining: exploring latent structure/representation of data

Figures here and in the next 7 slides from Stanford CS231n 2017 Lecture 13; also see Goodfellow et al., “Generative adversarial nets”, NIPS, 2014
Generative model (cont’)

What to learn from training data?
One way is to learn data’s density distribution from training data, then generate new data from the learned distribution.

Training data $\sim p_{\text{data}}(x)$
Generated samples $\sim p_{\text{model}}(x)$
Generative model (cont’)

- **What to learn from training data?**
  One way is to learn data’s density distribution from training data, then generate new data from the learned distribution.

- **Training data** \( \sim p_{data}(x) \)
- **Generated samples** \( \sim p_{model}(x) \)

- **Difficulty:** never know real data complex distribution \( P_{data}(x) \), therefore difficult to learn its approximation \( P_{model}(x) \)
Generative model (cont’)

- What to learn from training data?
  One way is to learn data’s density distribution from training data, then generate new data from the learned distribution.

- Difficulty: never know real data complex distribution $P_{data}(x)$, therefore difficult to learn its approximation $P_{model}(x)$

- Solution: learn to generate new data directly, without learning the data distribution!
Problem: How to generate data without learning complex high-dim (image) data distribution?
Generative model (cont’)

- Problem: How to generate data without learning complex high-dim (image) data distribution?
- Idea: sample from simple distribution (e.g., of random noise), and learn a complex transformation from simple distribution to the (unknown) complex distribution of image data.
Generative model (cont’)

- Problem: How to generate data without learning complex high-dim (image) data distribution?
- Idea: sample from simple distribution (e.g., of random noise), and learn a complex transformation from simple distribution to the (unknown) complex distribution of image data.
- If using a neural network to represent the transformation, the objective is to train the network to generate realistic images whose distribution is similar to that of training dataset.

![Diagram of Generative Model](image.png)
Generative adversarial network (GAN): a two-player game

- How to evaluate whether generated images have a similar distribution to the training set?
Generative adversarial network (GAN): a two-player game

- How to evaluate whether generated images have a similar distribution to the training set?
- Key insight: Use another network to help evaluate it!

Diagram:
- **Discriminator Network**
  - Accepts as input both real images (from the training set) and fake images (from the generator).
  - Decides if the input images are real or fake.

Diagram Nodes:
- **Random noise (z)**
- **Generator Network**
  - Generates synthetic images from the random noise input.
- **Fake Images (from generator)**
- **Real Images (from training set)**

Diagram Edges:
- From random noise (z) to the generator network.
- From the generator network to the discriminator network.
- From the discriminator network to real or fake.
Generative adversarial network (GAN): a two-player game

- How to evaluate whether generated images have a similar distribution to the training set?
- Key insight: Use another network to help evaluate it!

G network: try to fool D by generating realistic images
D network: try to distinguish between real and fake images
GAN training

- The two objectives can be achieved by

\[
\min_{G_{\theta}} \max_{D_w} \{ \mathbb{E}_{x \sim P_r} [\log D_w(x)] + \mathbb{E}_{z \sim P(z)} [\log(1 - D_w(G_{\theta}(z)))] \}
\]

where discriminator network \(D_w\) is a binary classifier.
The two objectives can be achieved by:

\[
\min_{G_{\theta}} \max_{D_{w}} \{ \mathbb{E}_{x \sim P_r} [\log D_w(x)] + \mathbb{E}_{z \sim P(z)} [\log (1 - D_w(G_{\theta}(z)))] \} 
\]

where discriminator network \(D_w\) is a binary classifier.

Train the two networks alternatively:

1: Training D network, i.e. \(\max_{D_w} \{ \cdot \}\), makes \(D_w(x)\) close to 1 for real images and \(D_w(G(z))\) close to 0 for fake images \(G(z)\)

\[
\max_{D_{w}} \{ \mathbb{E}_{x \sim P_r} [\log D_w(x)] + \mathbb{E}_{z \sim P(z)} [\log (1 - D_w(G_{\theta}(z)))] \} 
\]
GAN training

The two objectives can be achieved by

\[
\min_{G_{\theta}} \max_{D_w} \left\{ \mathbb{E}_{x \sim P_r} [\log D_w(x)] + \mathbb{E}_{z \sim P(z)} [\log (1 - D_w(G_{\theta}(z)))] \right\}
\]

where discriminator network \( D_w \) is a binary classifier.

Train the two networks alternatively:

1: Training D network, i.e. \( \max_{D_w} \{ \cdot \} \), makes \( D_w(x) \) close to 1 for real images and \( D_w(G(z)) \) close to 0 for fake images \( G(z) \)

\[
\max_{D_w} \left\{ \mathbb{E}_{x \sim P_r} [\log D_w(x)] + \mathbb{E}_{z \sim P(z)} [\log (1 - D_w(G_{\theta}(z)))] \right\}
\]

2: Training G network, i.e. \( \min_{G_{\theta}} \{ \cdot \} \), makes \( D_w(G(z)) \) close to 1, fooling D into thinking \( G(z) \) is real.

\[
\min_{G_{\theta}} \left\{ \mathbb{E}_{z \sim P(z)} [\log (1 - D_w(G_{\theta}(z)))] \right\}
\]
However, optimizing generator’s objective does not work well!

Figure: x-axis for $D(G(\tilde{z}))$ and y-axis for $\log(1 - D(G(\tilde{z})))$
GAN training (cont’)

Instead, optimizing generator by

$$\min_{G_\theta}\{\mathbb{E}_{z \sim P(z)} \left[ -\log(D_w(G_\theta(z))) \right]\}$$

which also tries to fool discriminator, but now with higher gradient signal for poor generator.
GAN implementation

for number of training iterations do
  for k steps do
    • Sample minibatch of \( m \) noise samples \( \{ z^{(1)}, \ldots, z^{(m)} \} \) from noise prior \( p_g(z) \).
    • Sample minibatch of \( m \) examples \( \{ x^{(1)}, \ldots, x^{(m)} \} \) from data generating distribution \( p_{data}(x) \).
    • Update the discriminator by ascending its stochastic gradient:
      \[
      \nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[ \log D_{\theta_d}(x^{(i)}) + \log(1 - D_{\theta_d}(G_{\theta_g}(z^{(i)}))) \right]
      \]
      Update Discriminator
  end for

  • Sample minibatch of \( m \) noise samples \( \{ z^{(1)}, \ldots, z^{(m)} \} \) from noise prior \( p_g(z) \).
  • Update the generator by ascending its stochastic gradient (improved objective):
    \[
    \nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log(D_{\theta_d}(G_{\theta_g}(z^{(i)})))
    \]
    Update Generator
  end for

• Note: \( D_{\theta_d} = D_w \) and \( G_{\theta_g} = G_{\theta} \)
- Generator (consisting of FC layers) generated realistic images
- Generator does not simply remember training images

Yellow boxes: nearest training example of neighboring generated sample

Figure from Goodfellow et al., “Generative adversarial nets”, NIPS, 2014
DCGAN: deep convolutional GAN

- Generator is a ‘deconvolutional’ network
- Discriminator is a convolutional network
- It is not that easy to make DCGAN work!

Architecture guidelines for stable Deep Convolutional GANs

- Replace any pooling layers with strided convolutions (discriminator) and fractional-strided convolutions (generator).
- Use batchnorm in both the generator and the discriminator.
- Remove fully connected hidden layers for deeper architectures.
- Use ReLU activation in generator for all layers except for the output, which uses Tanh.
- Use LeakyReLU activation in the discriminator for all layers.

Table here and figures in next 5 slides from Radford, Metz, Chintala, “Unsupervised representation learning with deep convolutional generative adversarial networks”, ICLR, 2016
DCGAN (cont’)

- The generator’s structure
DCGAN result

- Generated images are already realistic after one epoch training
Interpolations between random points in the latent (input) space result in semantic changes in generated images.
DCGAN result (cont’): interpretable vector arithmetic

In latent space:
mean smiling woman vector - mean neural woman vector + mean neural man vector = mean smiling man vector!
Similarly, women-with-glass images can be generated by vector arithmetic in the latent space.
LSGAN: Least Square Generative Adversarial Networks

- Uses least-square loss to relieve gradient vanishing issue. Original GAN used (sigmoid) binary cross-entropy loss.
- Tries to achieve the same objective as original GAN

Original GAN

\[
\min_G \max_D V_{GAN}(D, G) = \mathbb{E}_{x \sim p_{data}}(x) \left[ \log D(x) \right] + \mathbb{E}_{z \sim p_z}(z) \left[ \log(1 - D(G(z))) \right]
\]

LS-GAN

\[
\min_D V_{LSGAN}(D) = \frac{1}{2} \mathbb{E}_{x \sim p_{data}}(x) \left[ (D(x) - 1)^2 \right] + \frac{1}{2} \mathbb{E}_{z \sim p_z}(z) \left[ (D(G(z)))^2 \right]
\]

\[
\min_G V_{LSGAN}(G) = \frac{1}{2} \mathbb{E}_{z \sim p_z}(z) \left[ (D(G(z)) - 1)^2 \right].
\]

LSGAN result

- LSGAN generated good-quality (e.g., outdoor and indoor) images

![LSGAN images](image1)

- LSGAN is more stable than regular GAN, when batch normalization is removed in generator.

![LSGAN vs. Regular GAN](image2)
Issues in GAN training

- Unstable training: when Discriminator becomes perfect, loss becomes zero, so gradient for Generator vanishes!
Issues in GAN training

- Unstable training: when Discriminator becomes perfect, loss becomes zero, so gradient for Generator vanishes! On the other hand, if Discriminator behave badly, Generator would have bad feedback and cannot update well.

See theoretical proof behind the issues from Arjovsky, Bottou, "Towards principled methods for training generative adversarial networks", arXiv, 2017
Issues in GAN training

- Unstable training: when Discriminator becomes perfect, loss becomes zero, so gradient for Generator vanishes! On the other hand, if Discriminator behave badly, Generator would have bad feedback and cannot update well.

- Mode collapse: generator generates realistic images, but with low varieties

- Lack of proper evaluation metric: do not know when to stop training.

See theoretical proof behind the issues from Arjovsky, Bottou, “Towards principled methods for training generative adversarial networks”, arXiv, 2017
Essentially,...

- Generator is used to generate fake data whose distribution $P_\theta$ approximates real but unknown data distribution $P_r$.
- Discriminator helps measure the difference between $P_r$ and $P_\theta$ (using loss function of GAN based on discriminator's output).

Any better way to measure difference between distributions?
Distance measurements between two distributions

- Kullback-Leibler (KL) divergence

\[ KL(P_r \parallel P_\theta) = \int P_r(x) \log \left( \frac{P_r(x)}{P_\theta(x)} \right) dx \]
Distance measurements between two distributions

- Kullback-Leibler (KL) divergence

\[
KL(P_r \parallel P_\theta) = \int P_r(x) \log \left( \frac{P_r(x)}{P_\theta(x)} \right) dx
\]

- Jenson-Shannon (JS) divergence, with \( P_m = (P_r + P_\theta)/2 \):

\[
JS(P_r, P_\theta) = \frac{1}{2} KL(P_r \parallel P_m) + \frac{1}{2} KL(P_\theta \parallel P_m)
\]

(GAN’s loss function with optimal \( D \)) = \( 2JS(P_r, P_\theta) - 2 \log 2 \)
Distance measurements between two distributions

- Kullback-Leibler (KL) divergence

\[ KL(P_r \parallel P_\theta) = \int P_r(x) \log \left( \frac{P_r(x)}{P_\theta(x)} \right) dx \]

- Jenson-Shannon (JS) divergence, with \( P_m = (P_r + P_\theta)/2 \):

\[ JS(P_r, P_\theta) = \frac{1}{2} KL(P_r \parallel P_m) + \frac{1}{2} KL(P_\theta \parallel P_m) \]

(GAN’s loss function with optimal \( D \) = \( 2JS(P_r, P_\theta) - 2 \log 2 \))

- Earth Mover’s distance or Wasserstein distance:

\[ W(P_r, P_\theta) = \inf_{\gamma \in \Pi(P_r, P_\theta)} \mathbb{E}(x,y) \sim \gamma \left[ \| x - y \| \right] \]

where \( \Pi(P_r, P_\theta) \) is the set of all joint distributions whose marginal distributions are \( P_r \) and \( P_\theta \).
Earth Mover’s Distance (EMD)

- EMD is the minimal total amount of work it takes to transform one heap into the other.
- ‘work’: amount of moved earth in a chunk times the distance moved.
Earth Mover’s Distance (EMD)

- EMD is the minimal total amount of work it takes to transform one heap into the other.
- ‘work’: amount of moved earth in a chunk times the distance moved.

Figures from https://vincentherrmann.github.io/blog/wasserstein/
Example of measuring the distance between real distribution $P_0$ (i.e., $P_r$) and fake distribution $P_{\theta}$
EMD or Wasserstein distance (cont’)

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

EMD or Wasserstein distance (cont’)

- \( W(\mathbb{P}_0, \mathbb{P}_\theta) = |\theta| , \)

- \( JS(\mathbb{P}_0, \mathbb{P}_\theta) = \begin{cases} 
\log 2 & \text{if } \theta \neq 0 , \\
0 & \text{if } \theta = 0 , 
\end{cases} \)

- \( KL(\mathbb{P}_\theta || \mathbb{P}_0) = KL(\mathbb{P}_0 || \mathbb{P}_\theta) = \begin{cases} 
+\infty & \text{if } \theta \neq 0 , \\
0 & \text{if } \theta = 0 , 
\end{cases} \)

- The gradient of JS and KL divergence (over \( \theta \)) is always 0, therefore failing to update \( \theta \).

- Wasserstein distance is continuous and has non-zero gradient almost everywhere

- Therefore, JS and KL divergences are not good choices to measure the distance between two distributions.
It is intractable to compute Wasserstein distance exactly

\[ W(P_r, P_g) = \inf_{\gamma \in \Pi(P_r, P_g)} \mathbb{E}_{(x,y) \sim \gamma} [\|x - y\|] \]
Wasserstein distance

- It is intractable to compute Wasserstein distance exactly

\[ W(P_r, P_g) = \inf_{\gamma \in \Pi(P_r, P_g)} \mathbb{E}_{(x,y) \sim \gamma} [\|x - y\|] \]

- But it shows \( W \) is equivalent to

\[ W(P_r, P_\theta) = \sup_{\|f\|_L \leq 1} \mathbb{E}_{x \sim P_r} [f(x)] - \mathbb{E}_{x \sim P_\theta} [f(x)] \]

where \( \|f\|_L \leq 1 \) means \( f \) is a 1-Lipschitz function.
Wasserstein distance

- It is intractable to compute Wasserstein distance exactly

\[ W(P_r, P_g) = \inf_{\gamma \in \Pi(P_r, P_g)} \mathbb{E}_{(x,y) \sim \gamma} [\|x - y\|] \]

- But it shows \( W \) is equivalent to

\[ W(P_r, P_\theta) = \sup_{\|f\|_L \leq 1} \mathbb{E}_{x \sim P_r} [f(x)] - \mathbb{E}_{x \sim P_\theta} [f(x)] \]

where \( \|f\|_L \leq 1 \) means \( f \) is a 1-Lipschitz function.

- Let \( d_X \) and \( d_Y \) be distance functions on spaces \( X \) and \( Y \). A function \( f : X \rightarrow Y \) is K-Lipschitz if for all \( x_1, x_2 \in X \),

\[ d_Y(f(x_1), f(x_2)) \leq K d_X(x_1, x_2) \]

Intuitively: if \( x_1, x_2 \) are close, \( f(x_1) \) and \( f(x_2) \) are also close.
The supremum over 1-Lipschitz functions is still intractable

$$W(P_r, P_\theta) = \sup_{\|f\|_{L^1} \leq 1} \mathbb{E}_{x \sim P_r}[f(x)] - \mathbb{E}_{x \sim P_\theta}[f(x)]$$
Wasserstein distance approximation

- The supremum over 1-Lipschitz functions is still intractable

\[ W(P_r, P_\theta) = \sup_{\|f\|_{L} \leq 1} \mathbb{E}_{x \sim P_r} [f(x)] - \mathbb{E}_{x \sim P_\theta} [f(x)] \]

- Suppose there are a parameterized function family \( \{f_w\}_{w \in \mathcal{W}} \) with parameter \( w \) and function \( f_w \) being K-Lipschitz, then

\[
\begin{align*}
\max_{w \in \mathcal{W}} \mathbb{E}_{x \sim P_r} [f_w(x)] - \mathbb{E}_{x \sim P_\theta} [f_w(x)] & \leq \sup_{\|f\|_{L} \leq K} \mathbb{E}_{x \sim P_r} [f(x)] - \mathbb{E}_{x \sim P_\theta} [f(x)] \\
\end{align*}
\]

Note: K can be absorbed into learning rate during max.
The supremum over 1-Lipschitz functions is still intractable

\[ W(P_r, P_\theta) = \sup_{\|f\|_{L} \leq 1} \mathbb{E}_{x \sim P_r} [f(x)] - \mathbb{E}_{x \sim P_\theta} [f(x)] \]

Suppose there are a parameterized function family \( \{f_w\}_{w \in \mathcal{W}} \) with parameter \( w \) and function \( f_w \) being K-Lipschitz, then

\[
\begin{align*}
\max_{w \in \mathcal{W}} \mathbb{E}_{x \sim P_r} [f_w(x)] - \mathbb{E}_{x \sim P_\theta} [f_w(x)] & \leq \sup_{\|f\|_{L} \leq K} \mathbb{E}_{x \sim P_r} [f(x)] - \mathbb{E}_{x \sim P_\theta} [f(x)] \\
& = K \cdot W(P_r, P_\theta)
\end{align*}
\]
Wasserstein distance approximation

- The supremum over 1-Lipschitz functions is still intractable
  \[ W(P_r, P_\theta) = \sup_{\|f\|_L \leq 1} \mathbb{E}_{x \sim P_r} [f(x)] - \mathbb{E}_{x \sim P_\theta} [f(x)] \]

- Suppose there are a parameterized function family \( \{f_w\}_{w \in \mathcal{W}} \) with parameter \( w \) and function \( f_w \) being K-Lipschitz, then
  \[
  \max_{w \in \mathcal{W}} \mathbb{E}_{x \sim P_r} [f_w(x)] - \mathbb{E}_{x \sim P_\theta} [f_w(x)] \\
  \leq \sup_{\|f\|_L \leq K} \mathbb{E}_{x \sim P_r} [f(x)] - \mathbb{E}_{x \sim P_\theta} [f(x)] \\
  = K \cdot W(P_r, P_\theta)
  \]

- Then \( W(P_r, P_\theta) \) can be approximated by
  \[
  \max_{w \in \mathcal{W}} \mathbb{E}_{x \sim P_r} [f_w(x)] - \mathbb{E}_{x \sim P_\theta} [f_w(x)]
  \]

  Note: \( K \) can be absorbed into learning rate during max.
Wasserstein GAN

- $f_w$ can be represented by a neural network, with parameter $w$
Wasserstein GAN

- $f_w$ can be represented by a neural network, with parameter $w$
- But: how to make sure $f_w$ is K-Lipschitz?
Wasserstein GAN

- $f_w$ can be represented by a neural network, with parameter $w$
- But: how to make sure $f_w$ is K-Lipschitz?
- Answer: weight clamping! Clipping $w$ to be within $[-c, c]$ after every update of $w$. 

Go back to original objective: In order to train Generator $g_\theta$ such that the fake $P_\theta = g_\theta(Z)$ matches real $P_r$, we need a distance measurement to estimate the difference between $P_\theta$ and $P_r$. Here the distance measurement is Wasserstein distance which can be approximately computed based on the optimal $f_w$, i.e., by optimizing a network work $f_w$. We call such a network $f_w$‘critic function’, corresponding to Discriminator in original GAN.

Wasserstein GAN: generator + critic
Wasserstein GAN

- $f_w$ can be represented by a neural network, with parameter $w$
- But: how to make sure $f_w$ is K-Lipschitz?
- Answer: weight clamping! Clipping $w$ to be within $[-c, c]$ after every update of $w$.
- Go back to original objective: In order to train Generator $g_\theta$ such that the fake $P_\theta = g_\theta(Z)$ matches real $P_r$, we need a distance measurement to estimate the difference between $P_\theta$ and $P_r$. Here the distance measurement is Wasserstein distance which can be approximately computed based on the optimal $f_w$, i.e., by optimizing a network work $f_w$. We call such a network $f_w$ ‘critic function’, corresponding to Discriminator in original GAN.
Wasserstein GAN

- $f_w$ can be represented by a neural network, with parameter $w$
- But: how to make sure $f_w$ is K-Lipschitz?
- Answer: weight clamping! Clipping $w$ to be within $[-c, c]$ after every update of $w$.
- Go back to original objective: In order to train Generator $g_\theta$ such that the fake $P_\theta = g_\theta(Z)$ matches real $P_r$, we need a distance measurement to estimate the difference between $P_\theta$ and $P_r$. Here the distance measurement is Wasserstein distance which can be approximately computed based on the optimal $f_w$, i.e., by optimizing a network work $f_w$. We call such a network $f_w$ ‘critic function’, corresponding to Discriminator in original GAN.
- Wasserstein GAN: generator + critic
Once getting optimal $f_w$ (by training critic network with maximization of $\mathbb{E}_{x \sim P_r} [f_w(x)] - \mathbb{E}_{x \sim P_\theta} [f_w(x)]$), then

$$W(P_r, P_\theta) = \mathbb{E}_{x \sim P_r} [f_w(x)] - \mathbb{E}_{x \sim P_\theta} [f_w(x)]$$
Wasserstein GAN (cont’)

• Once getting optimal \( f_w \) (by training critic network with maximization of \( \mathbb{E}_{x \sim P_r} [f_w(x)] - \mathbb{E}_{x \sim P_\theta} [f_w(x)] \)), then

\[
W(P_r, P_\theta) = \mathbb{E}_{x \sim P_r} [f_w(x)] - \mathbb{E}_{x \sim P_\theta} [f_w(x)]
\]

• Then update generator \( g_\theta \) (with fixed \( f_w \)) by minimizing \( W \)

\[
\nabla_\theta W(P_r, P_\theta) = \nabla_\theta (\mathbb{E}_{x \sim P_r} [f_w(x)] - \mathbb{E}_{z \sim Z} [f_w(g_\theta(z))])

= -\mathbb{E}_{z \sim Z} [\nabla_\theta f_w(g_\theta(z))] 
\]
Wasserstein GAN (cont’)

- Once getting optimal $f_w$ (by training critic network with maximization of $\mathbb{E}_{x \sim P_r}[f_w(x)] - \mathbb{E}_{x \sim P_\theta}[f_w(x)]$), then
  \[
  W(P_r, P_\theta) = \mathbb{E}_{x \sim P_r}[f_w(x)] - \mathbb{E}_{x \sim P_\theta}[f_w(x)]
  \]

- Then update generator $g_\theta$ (with fixed $f_w$) by minimizing $W$
  \[
  \nabla_\theta W(P_r, P_\theta) = \nabla_\theta (\mathbb{E}_{x \sim P_r}[f_w(x)] - \mathbb{E}_{z \sim Z}[f_w(g_\theta(z))])
  = -\mathbb{E}_{z \sim Z}[\nabla_\theta f_w(g_\theta(z))]
  \]

In summary, repeat the following steps to train WGAN:

**Step 1:** Fix generator $g_\theta$, compute approximation of $W(P_r, P_\theta)$ by training $f_w$, i.e., by $\max_{w \in W} \mathbb{E}_{x \sim P_r}[f_w(x)] - \mathbb{E}_{x \sim P_\theta}[f_w(x)]$

**Step 2:** Fix ‘critic’ $f_w$, update $g_\theta$ with gradient $-\mathbb{E}_{z \sim Z}[\nabla_\theta f_w(g_\theta(z))]$ by sampling several $z$ from uniform distribution $Z$. 
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:** : $\alpha$, the learning rate. $c$, the clipping parameter. $m$, the batch size. $n_{\text{critic}}$, the number of iterations of the critic per generator iteration.

**Require:** : $w_0$, initial critic parameters. $\theta_0$, initial generator’s parameters.

1: while $\theta$ has not converged do
2:   for $t = 0, \ldots, n_{\text{critic}}$ 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
WGAN results

- Wasserstein loss correlates well with image quality
WGAN results (cont’)

- WGAN (top 2 rows; with same DCGAN structure) performs well (if not better) compared to DCGAN (bottom 2 rows)
WGAN results (cont’)

- WGAN (top 2 rows; with same DCGAN structure) still performs well when removing batch normalization, but DCGAN not (bottom 2 rows)
WGAN results (cont’)

- WGAN (top 2 rows; with same MLP structure) reduce issue of mode collapse, compared to original GAN (bottom 2 rows).
However, ... 

Issues of weight clipping in WGAN:
- makes critic $f_w$ within a small subset of K-Lipschitz functions.
- causes gradient vanishing or exploding (Fig. left).
- pushes weights to extremes of clipping range (Fig. right), in turn causing gradient exploding and slow training.

Figures here and in the next 3 slides from Gulrajani, Ahmed, Arjovsky, Dumoulin, Courville, “Improved training of Wasserstein GANs”, arXiv, 2017
Alternative way to enforce Lipschitz constraint: a function is 1-Lipschitz if and only if it has gradients (over input variable) with norm less or equal to 1.0 everywhere.
WGAN-GP: GAN with gradient penalty

- Alternative way to enforce Lipschitz constraint: a function is 1-Lipschitz if and only if it has gradients (over input variable) with norm less or equal to 1.0 everywhere.

- Critic loss (to be minimized)

\[
L = -\mathbb{E}_{x \sim P_r} [f_w(x)] + \mathbb{E}_{\tilde{x} \sim P_\theta} [f_w(\tilde{x})] + \lambda \mathbb{E}_{\hat{x} \sim P_{\hat{x}}} [(\|\nabla_{\hat{x}} f_w(\hat{x})\|_2 - 1)^2]
\]

where the last term is ‘gradient penalty’ term, and

\[
\epsilon \sim U[0, 1], x \sim P_r, \tilde{x} \in P_\theta
\]

\[
\hat{x} = \epsilon x + (1 - \epsilon)\tilde{x}
\]

considering optimal critic has gradient norm 1.0 on straight lines connecting points from \( P_r \) to \( P_\theta \).

- The improved WGAN is called WGAN-GP
• WGAN-GP improves training speed compared to WGAN

Convergence on CIFAR-10

• ‘Inception score’ is used to measure image quality
WGAN-GP result

- WGAN-GP successfully train difficult GAN architectures

<table>
<thead>
<tr>
<th>DCGAN</th>
<th>LSGAN</th>
<th>WGAN (clipping)</th>
<th>WGAN-GP</th>
</tr>
</thead>
<tbody>
<tr>
<td>tanh nonlinearities everywhere in $G$ and $D$</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

101-layer ResNet $G$ and $D$
Summary

- GANs provide an effective way to generate data
- GANs do not explicitly model data distribution
- GAN training: unstable and mode collapse
- WGAN: 1st attempt to improve GAN with theoretical proof
- More GANs already proposed, more GANs to be proposed!

Further reading:

- Miyato et al., Spectral normalization for generative adversarial networks, ICLR, 2018
- Dai et al., Good semi-supervised learning that requires a bad GAN, arXiv, 2017