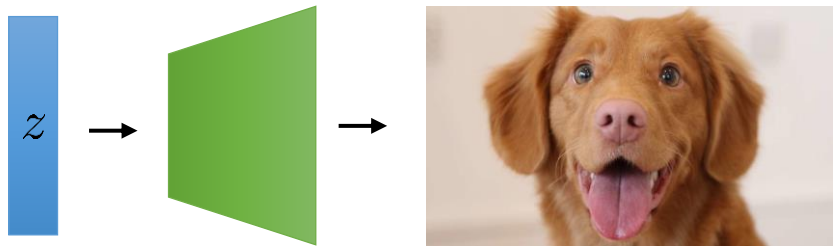
Generative Adversarial Networks

Designing, Visualizing and Understanding Deep Neural Networks

CS W182/282A

Instructor: Sergey Levine
UC Berkeley

Back to latent variable models



Idea: instead of training an encoder, can we just train the whole model to generate images that look similar to real images *at the population level*?

Using the model for **generation**:

1. sample $z \sim p(z)$ "generate a vector of random numbers"
2. sample $x \sim p(x|z)$ "turn that vector of random numbers into an image"

Matching distributions
at the population level

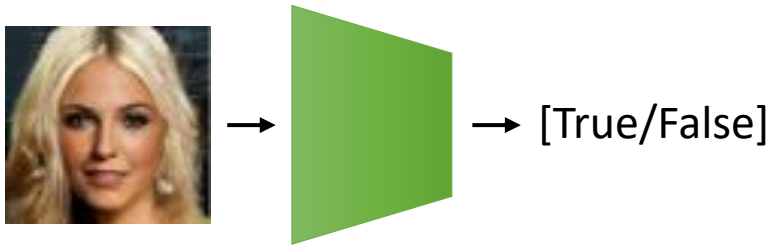
no two faces are the same, but they look similar **at the population level**

which set of faces is real?

it's a trick question…

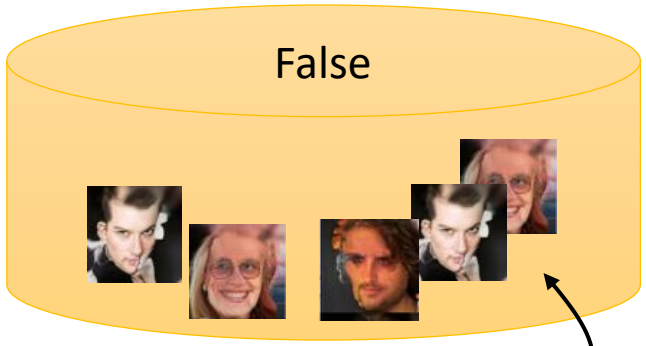
The only winning move is to generate

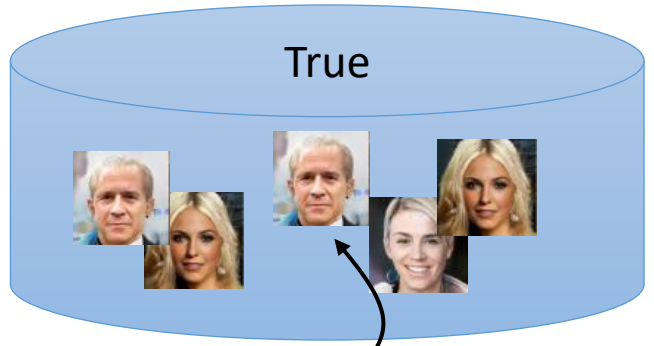
Idea: train a **network** to guess which images are real and which are fake!

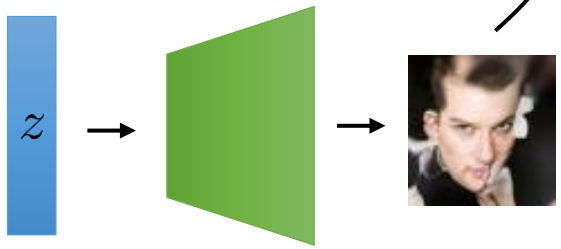


"is this a **real** image"

This model can then serve as a loss function for the generator!







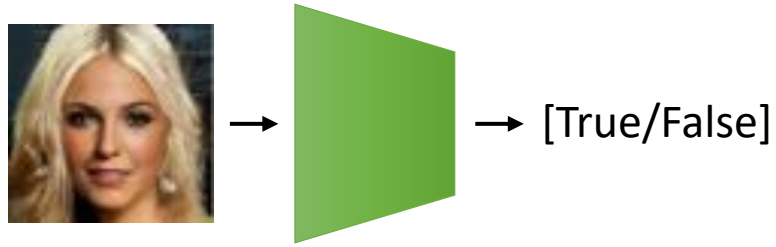
$p(z)$ $p_\theta(x|z)$

The only winning move is to generate

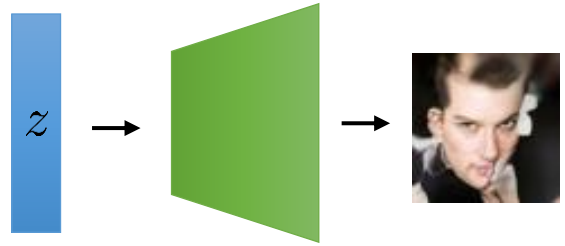
1. get a "True" dataset $\mathcal{D}_T = \{(x_i)\}$
2. get a generator $G_\theta(z)$ (how?)
3. sample a "False" dataset $\mathcal{D}_F$: $z \sim p(z)$, $x = G(z)$
4. train a discriminator $D_\phi(x) = p_\phi(y|x)$ using $\mathcal{D}_T$ and $\mathcal{D}_F$
5. use $-\log D(x)$ as "loss" to train $G(z)$

if only done once, too easy for $G(z)$ to "fool" $D(x)$

this **almost** works, but has two major problems



"discriminator"



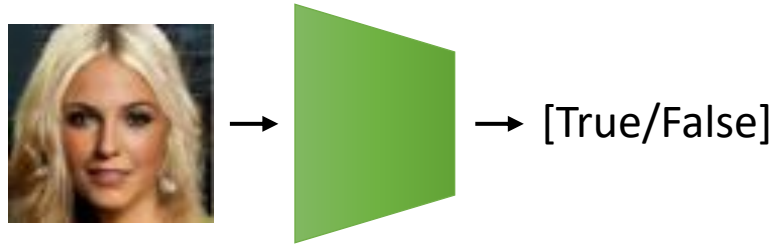
"generator"

The only winning move is to generate

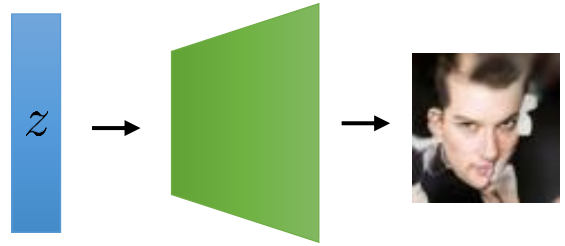
1. get a "True" dataset $\mathcal{D}_T = \{(x_i)\}$
2. get a generator $G_\theta(z)$ random initialization!
3. sample a "False" dataset $\mathcal{D}_F$: $z \sim p(z)$, $x = G(z)$
4. update $D_\phi(x) = p_\phi(y|x)$ using $\mathcal{D}_T$ and $\mathcal{D}_F$ (1 SGD step)
5. use $-\log D(x)$ as "loss" to update $G(z)$ (1 SGD step)

(in reality there are a variety of different losses, but similar idea…)

this is called a **generative adversarial network (GAN)**



"discriminator"

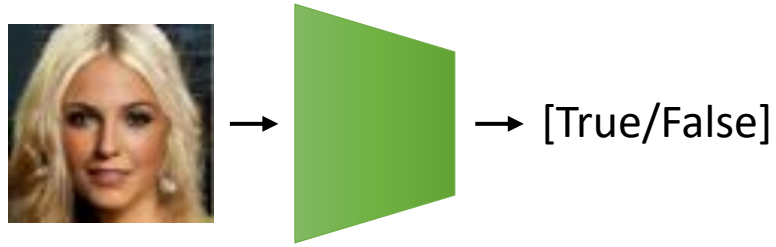


"generator"

Why do GANs learn **distributions**?

1. get a "True" dataset $\mathcal{D}_T = \{(x_i)\}$
2. get a generator $G_\theta(z)$ random initialization!
3. sample a "False" dataset $\mathcal{D}_F$: $z \sim p(z)$, $x = G(z)$
4. update $D_\phi(x) = p_\phi(y|x)$ using $\mathcal{D}_T$ and $\mathcal{D}_F$ (1 SGD step)
5. use $-\log D(x)$ as "loss" to update $G(z)$ (1 SGD step)

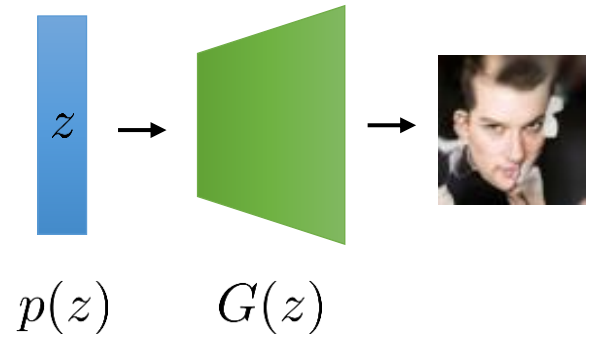
(in reality there are a variety of different losses, but similar idea…)



"discriminator"

what does $G(z)$ want to do?

make $D(x) = 0.5$ for all generated x "can't tell if real or fake"



"generator"

How to do this?

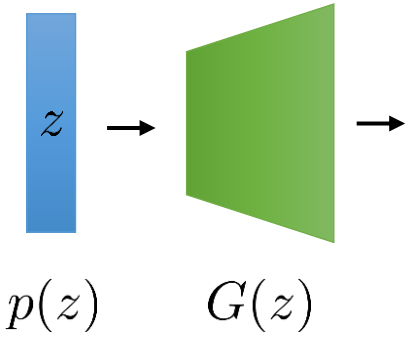
- Generate images that look realistic (obviously)
- Generate **all** possible realistic images why?

Why do GANs learn **distributions**?

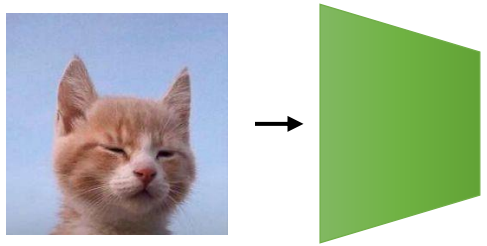
- Generate **all** possible realistic images

The generator will do **better** if it not only generates realistic pictures, but if it generates **all** realistic pictures

why?



very realistic, but only dogs



→ True = 1.0!

→ True = 0.25!

Small GANs

real pictures

Goodfellow et al. **Generative adversarial networks**. 2014

High-res GANs

Figure 5: 1024 × 1024 images generated using the CELEBA-HQ dataset. See Appendix F for a larger set of results, and the accompanying video for latent space interpolations.

Mao et al. (2016b) (128 × 128) Gulrajani et al. (2017) (128 × 128) Our (256 × 256)

Karras et al. **Progressive Growing of GANs**. 2017

Big GANs

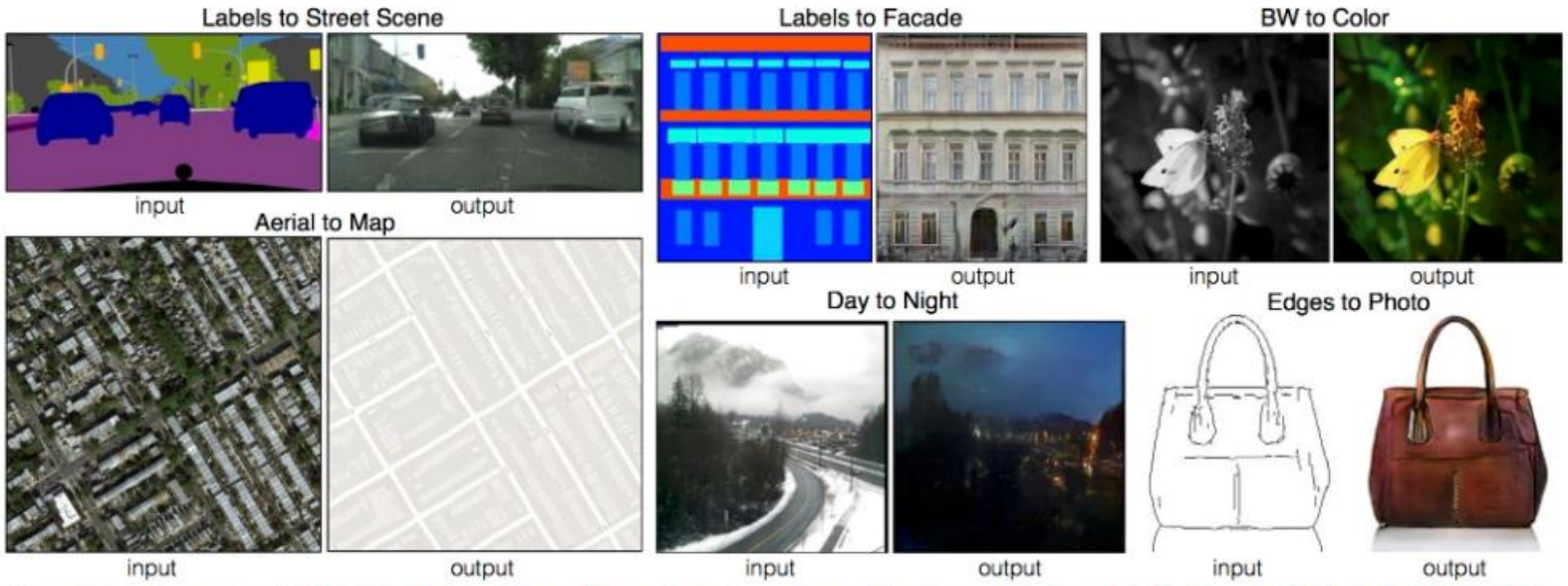
Figure 1: Class-conditional samples generated by our model.

(a) 128×128 (b) 256×256 (c) 512×512 (d)

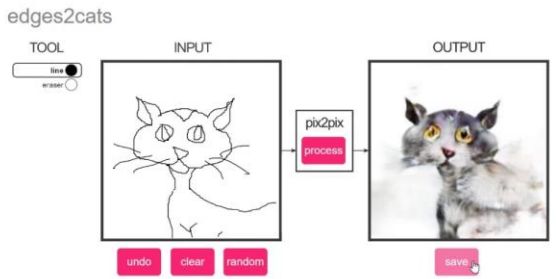
Figure 4: Samples from our BigGAN model with truncation threshold 0.5 (a-c) and an example of class leakage in a partially trained model (d).

Brock et al. **Large-Scale GAN Training…**. 2018

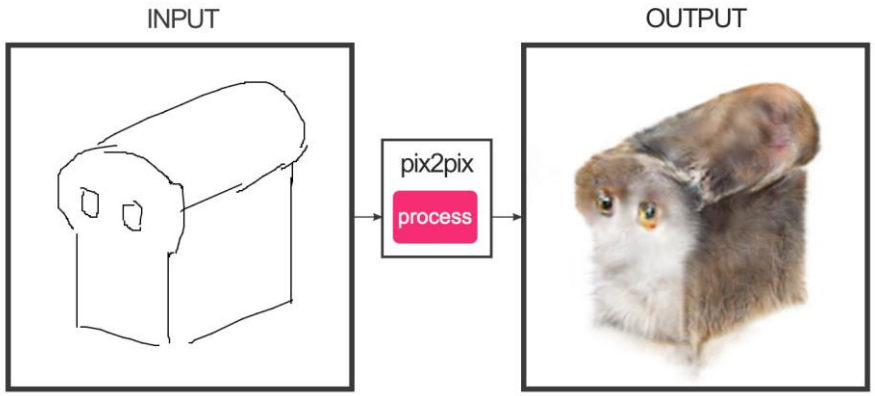
Turning bread into cat…



Example results on several image-to-image translation problems. In each case we use the same architecture and objective, simply training on different data.



Trained on about 2k stock cat photos and edges automatically generated from those photos. Generates cat-colored objects, some with nightmare faces. The best one I've seen yet was a cat-beholder.

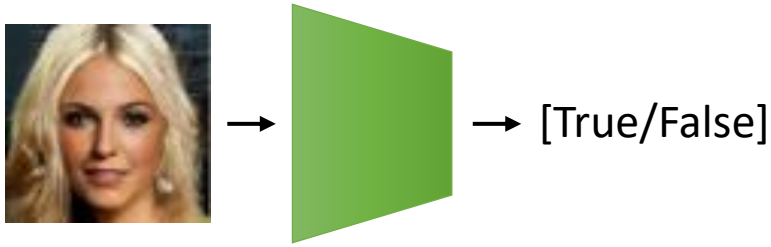


Isola et al. **Image-to-Image Translation with Conditional Adversarial Nets**. 2017

Generative Adversarial Networks

The GAN game

1. get a "True" dataset $\mathcal{D}_T = \{(x_i)\}$
2. get a generator $G_\theta(z)$ random initialization!
3. sample a "False" dataset $\mathcal{D}_F$: $z \sim p(z)$, $x = G(z)$
4. update $D_\phi(x) = p_\phi(y|x)$ using $\mathcal{D}_T$ and $\mathcal{D}_F$ (1 SGD step)
5. use $D(x)$ to update $G(z)$ (1 SGD step)



"discriminator"

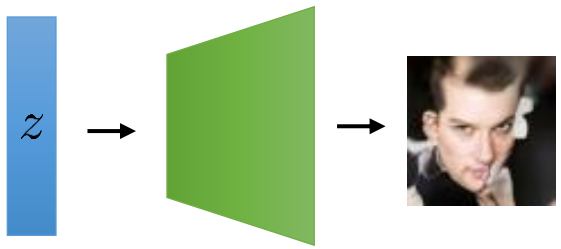
"classic" GAN 2-player game:

$$\min_G \max_D V(D,G) = E_{x\sim p_{\text{data}}(x)}[\log D(x)] + E_{z\sim p(z)}[\log(1 - D(G(z)))]$$

$$\approx \frac{1}{N}\sum_{i=1}^{N} \log D(x_i) \quad x_i \in \mathcal{D}_T \qquad \approx \frac{1}{N}\sum_{j=1}^{N} \log(1 - D(x_j))$$

$$x_j = G(z_j)$$

random numbers



$p(z)$ $G(z)$

"generator"

The GAN game

$$\min_{\theta} \max_{\phi} V(\theta, \phi) = E_{x \sim p_{\text{data}}(x)}[\log D_\phi(x)] + E_{z \sim p(z)}[\log(1 - D_\phi(G_\theta(z)))]$$

$$\phi \leftarrow \phi + \alpha \nabla_\phi V(\theta, \phi) \approx \nabla_\phi \left(\frac{1}{N} \sum_{i=1}^{N} \log D_\phi(x_i) + \frac{1}{N} \sum_{j=1}^{N} \log(1 - D_\phi(x_j)) \right)$$

$$x_i \in \mathcal{D}_T \qquad x_j = G(z_j)$$

this is just cross-entropy loss!

$$\theta \leftarrow \theta - \alpha \nabla_\theta V(\theta, \phi)$$

$$\approx \nabla_\theta \left(\frac{1}{N} \sum_{j=1}^{N} \log(1 - D_\phi(G_\theta(z_j))) \right)$$

random numbers

Two important details:

How to make this work with **stochastic** gradient descent/ascent?

How to compute the gradients?

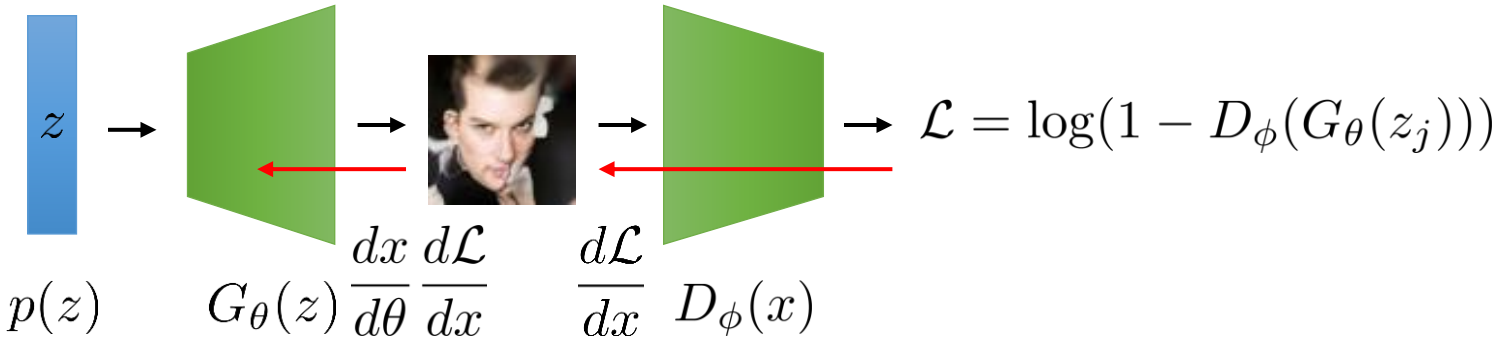
(both are actually pretty simple)

The GAN game

$$\min_{\theta} \max_{\phi} V(\theta, \phi) = E_{x \sim p_{\text{data}}(x)}[\log D_\phi(x)] + E_{z \sim p(z)}[\log(1 - D_\phi(G_\theta(z)))]$$

$$\phi \leftarrow \phi + \alpha \nabla_\phi V(\theta, \phi)$$
$$\theta \leftarrow \theta - \alpha \nabla_\theta V(\theta, \phi)$$

$$\nabla_\theta \left(\frac{1}{N} \sum_{j=1}^{N} \log(1 - D_\phi(G_\theta(z_j))) \right)$$



Just backpropagate from the discriminator into the generator!

What does the GAN optimize?

$$\min_G \max_D V(D,G) = E_{x\sim p_{\text{data}}(x)}[\log D(x)] + E_{z\sim p(z)}[\log(1 - D(G(z)))]$$

what can we say about $G(z)$ *at convergence*?

idea: express $D(x)$ *in closed form* as function of $G(z)$

$$D^\star_G(x) = \frac{p_{\text{data}}(x)}{p_{\text{data}}(x) + p_G(x)}$$

$$x = G(z) \quad z \sim p(z)$$

now what is the objective for G?

$V(D^\star_G, G) =$ ⟵ entirely a function of G

$E_{p_{\text{data}}(x)}[\log p_{\text{data}}(x) - \log(p_{\text{data}}(x) + p_G(x))] +$

$E_{p_G(x)}[\log p_G(x) - \log(p_{\text{data}}(x) + p_G(x))]$

$\mathcal{D}_T = \{x_i \sim p(x)\} \quad \mathcal{D}_F = \{x_j \sim q(x)\}$

$D^\star = \arg\max_D E_p[\log D(x)] + E_q[\log(1 - D(x))]$

$$\nabla_D = E_p\left[\frac{1}{D(x)}\right] - E_q\left[\frac{1}{1 - D(x)}\right] = 0$$

plug in $D^\star(x) = \frac{p(x)}{p(x) + q(x)}$ optimal discriminator

$$\sum_x \cancel{p(x)} \frac{p(x) + q(x)}{\cancel{p(x)}} - \sum_x \cancel{q(x)} \frac{p(x) + q(x)}{\cancel{q(x)}} = 0$$

What does the GAN optimize?

$$V(D_G^\star, G) =$$

$$E_{p_{\text{data}}(x)}[\log p_{\text{data}}(x) - \log(p_{\text{data}}(x) + p_G(x))] +$$

$$E_{p_G(x)}[\log p_G(x) - \log(p_{\text{data}}(x) + p_G(x))]$$

what funny expressions…

$$\text{let } q(x) = \frac{p_{\text{data}}(x) + p_G(x)}{2}$$

accounts for the $\frac{1}{2}$ factors

$$V(D_G^\star, G) = E_{p_{\text{data}}(x)}[\log p_{\text{data}}(x) - \log q(x)] + E_{p_G(x)}[\log p_G(x) - \log q(x)] - \log 4$$

$$D_{\text{KL}}(p_{\text{data}} \| q(x)) \qquad D_{\text{KL}}(p_G \| q(x))$$

$$= D_{\text{JS}}(p_{\text{data}} \| p_G)$$

Jensen-Shannon divergence

This has some interesting properties:

- goes to zero if the distributions match
- symmetric (unlike KL-divergence)

This means the GAN really is trying to match the data distribution!

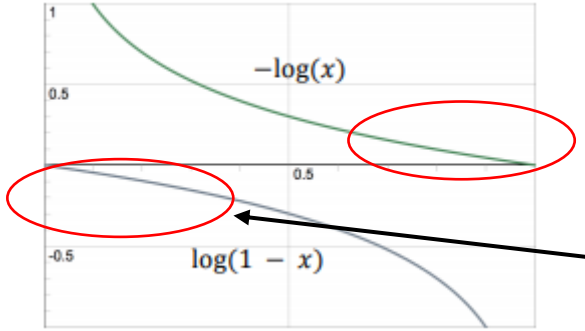
A small practical aside

$$\min_{\theta} \max_{\phi} V(\theta, \phi) = E_{x \sim p_{\text{data}}(x)}[\log D_\phi(x)] + E_{z \sim p(z)}[\log(1 - D_\phi(G_\theta(z)))]$$

generator loss should be $E_{z \sim p(z)}[\log(1 - D_\phi(G_\theta(z)))]$ "minimize probability that image is fake"

in practice, we often use $E_{z \sim p(z)}[-\log D_\phi(G_\theta(z))]$ "maximize probability that image is real"

(though there are other variants too!)

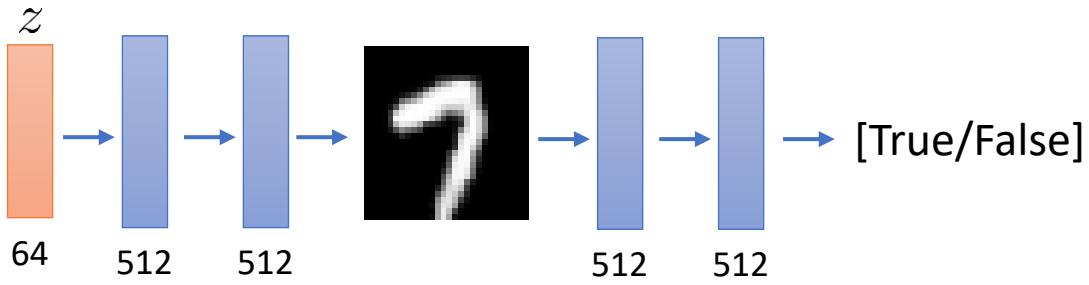


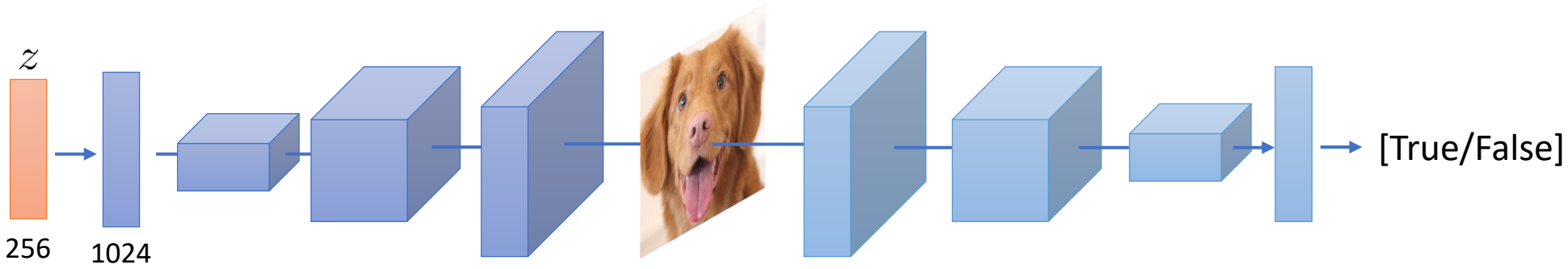
small gradient when generator is good
(i.e., probability of real is high)

small gradient when generator is bad
(i.e., probability of fake is high)

GAN architectures

some made-up architectures





transpose convolutions

a real architecture (BigGAN)

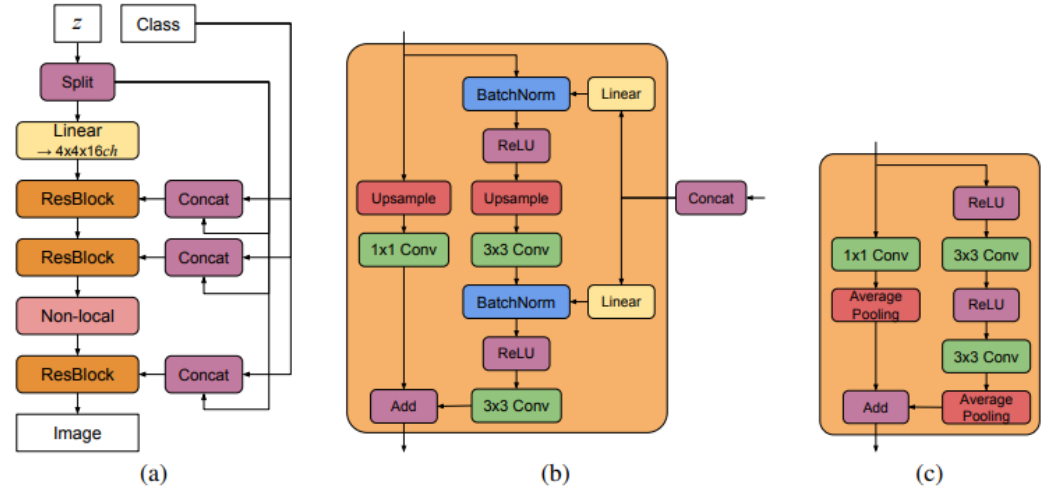
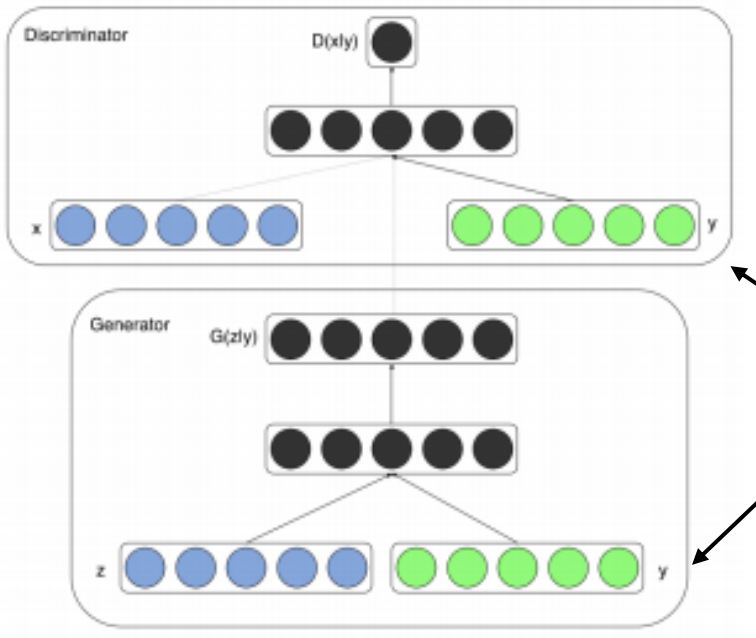


Figure 15: (a) A typical architectural layout for BigGAN's **G**; details are in the following tables. (b) A Residual Block (*ResBlock up*) in BigGAN's **G**. (c) A Residual Block (*ResBlock down*) in BigGAN's **D**.

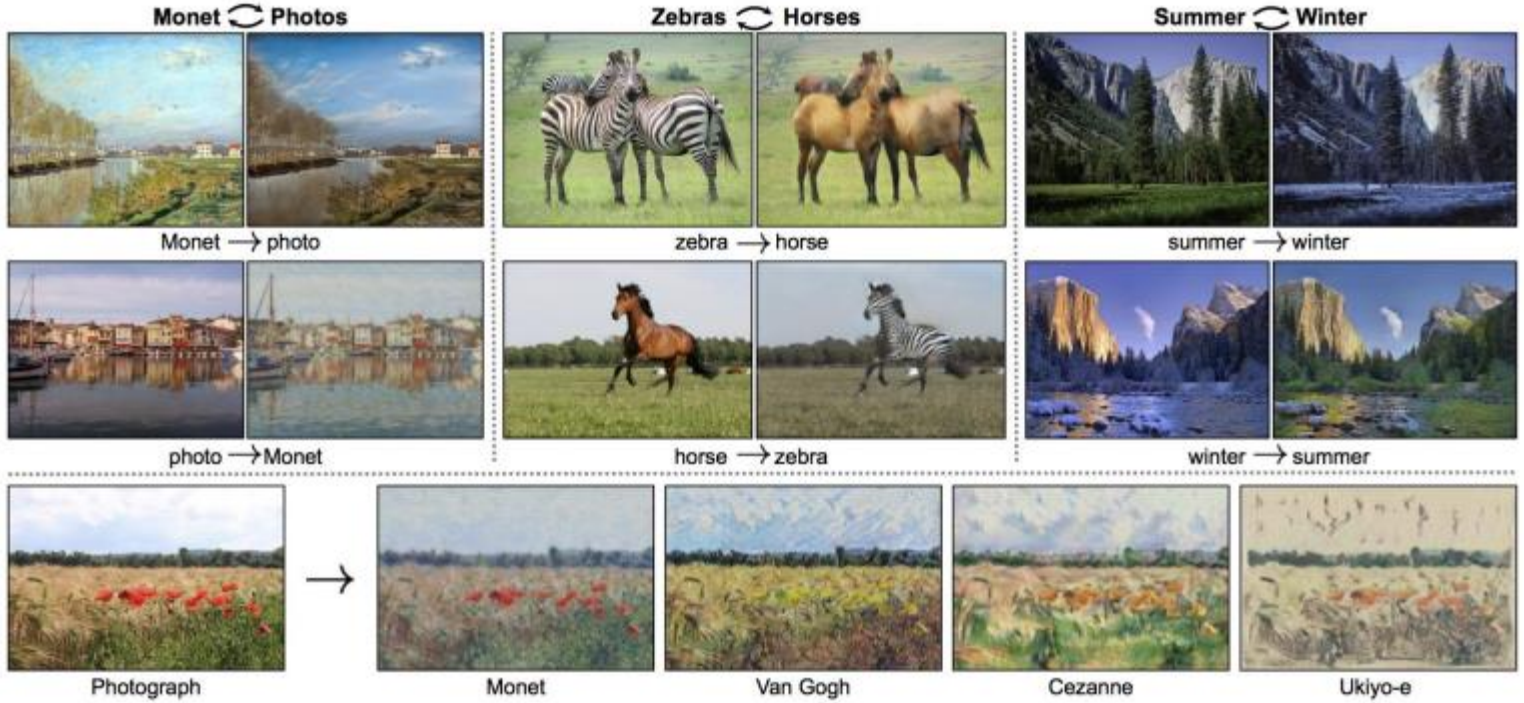
Conditional GANs



append conditioning (e.g., class label)
to **both** generator and discriminator

Mirza, M., & Osindero, S. (2014). Conditional generative adversarial nets.

CycleGAN



Problem: we don't know which images "go together"

Zhu, J. Y., Park, T., Isola, P., & Efros, A. A. (2017). **Unpaired image-to-image translation using cycle-consistent adversarial networks**.

CycleGAN

two (conditional) generators:

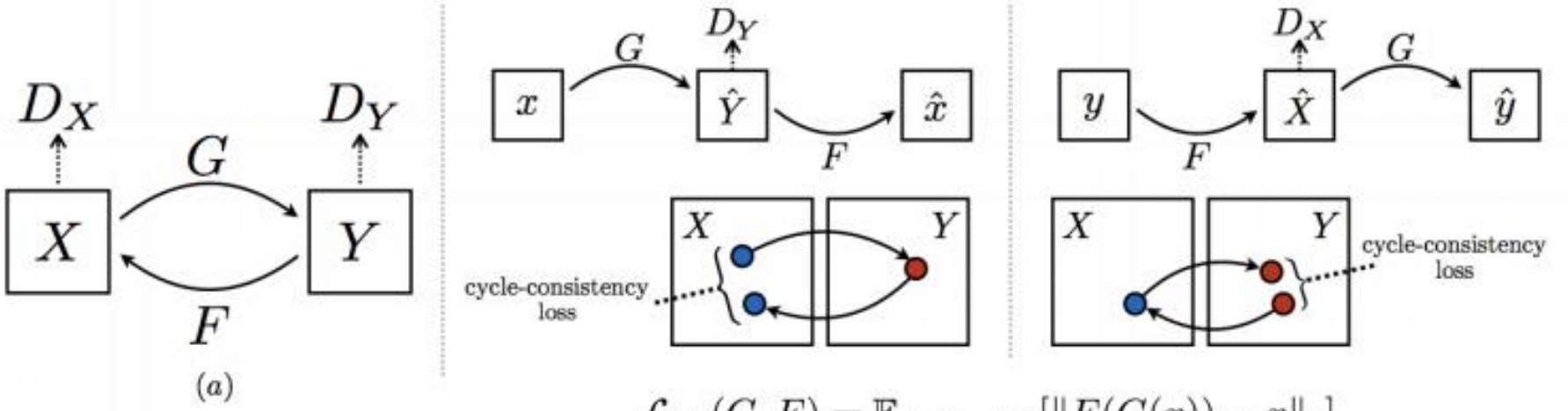
G: turn X into Y (e.g., horse into zebra)

F: turn Y into X (e.g., zebra into horse)

two discriminators:

D_X: is it a realistic horse?

D_Y: is it a realistic zebra?



$$\mathcal{L}_{\text{cyc}}(G, F) = \mathbb{E}_{x \sim p_{\text{data}}(x)}[\|F(G(x)) - x\|_1] + \mathbb{E}_{y \sim p_{\text{data}}(y)}[\|G(F(y)) - y\|_1].$$

If I turn this horse into a zebra, and then turn that zebra back into a horse, I should get the same horse!

Problem: why should the "translated" zebra look anything like the original horse?

Zhu, J. Y., Park, T., Isola, P., & Efros, A. A. (2017). **Unpaired image-to-image translation using cycle-consistent adversarial networks**.

Summary

- The GAN is a 2-player game $\min_G \max_D V(D,G) = E_{x\sim p_{\text{data}}(x)}[\log D(x)] + E_{z\sim p(z)}[\log(1 - D(G(z)))]$

- We can derive the **optimal** discriminator, and from this determine what objective is minimized at the Nash equilibrium
  - Note that this does **not** guarantee that we'll actually find the Nash equilibrium!

$$D_G^\star(x) = \frac{p_{\text{data}}(x)}{p_{\text{data}}(x) + p_G(x)}$$

$$V(D_G^\star, G) = D_{\text{JS}}(p_{\text{data}} \| p_G)$$

- We can train either fully connected **or** convolutional GANs

- We can turn horses into zebras

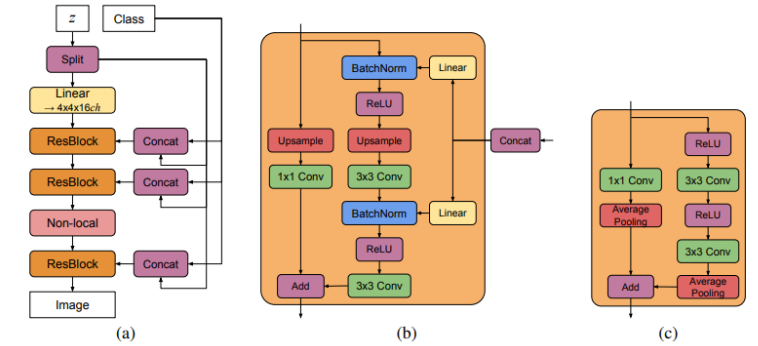
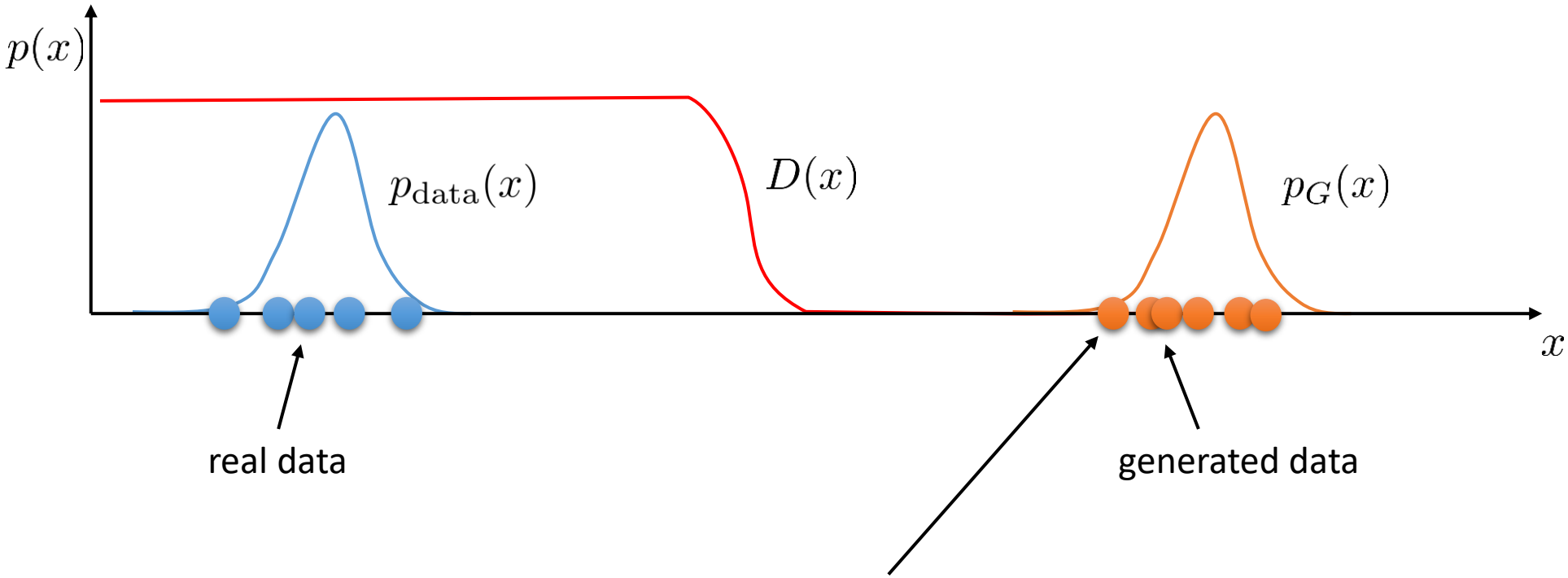


Figure 15: (a) A typical architectural layout for BigGAN's **G**; details are in the following tables. (b) A Residual Block (*ResBlock up*) in BigGAN's **G**. (c) A Residual Block (*ResBlock down*) in BigGAN's **D**.

Improved GAN Training

Why is training GANs hard?

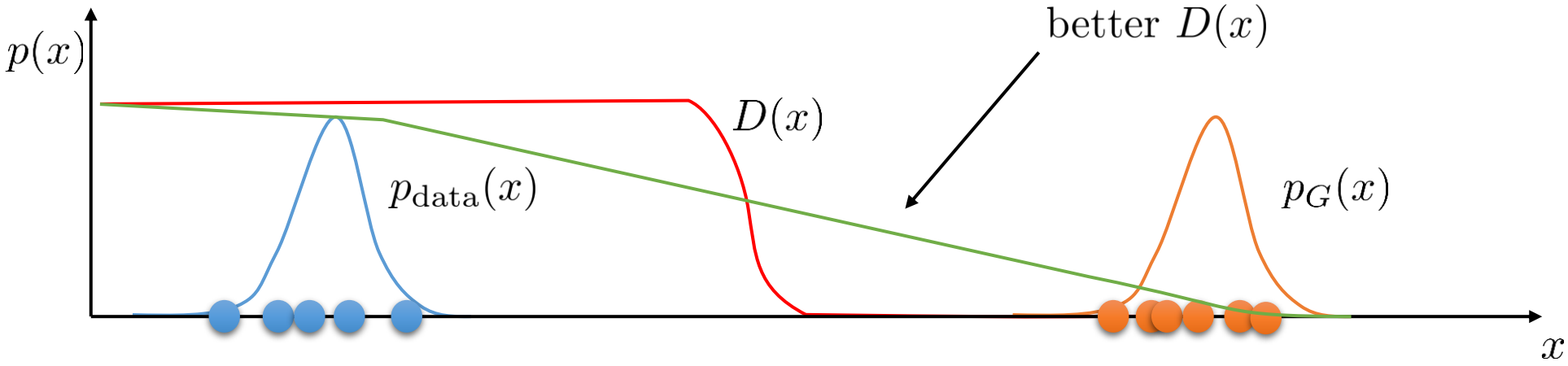


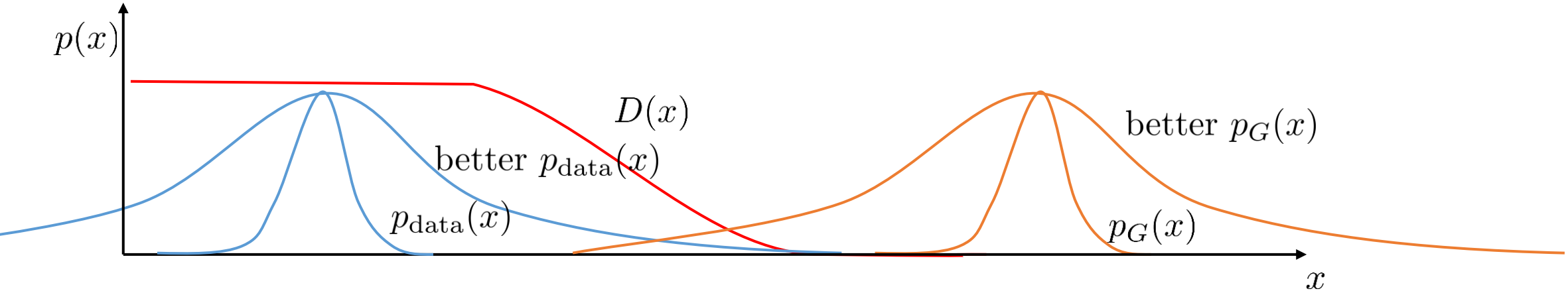
what is the generator gradient here?

$$\min_{\theta} \max_{\phi} V(\theta, \phi) = E_{x \sim p_{\text{data}}(x)}[\log D_\phi(x)] + E_{z \sim p(z)}[\log(1 - D_\phi(G_\theta(z)))]$$

all of these values are basically the same

How can we make it better?





Improved GAN techniques

(in no particular order)

Least-squares GAN (LSGAN) discriminator outputs real-valued number

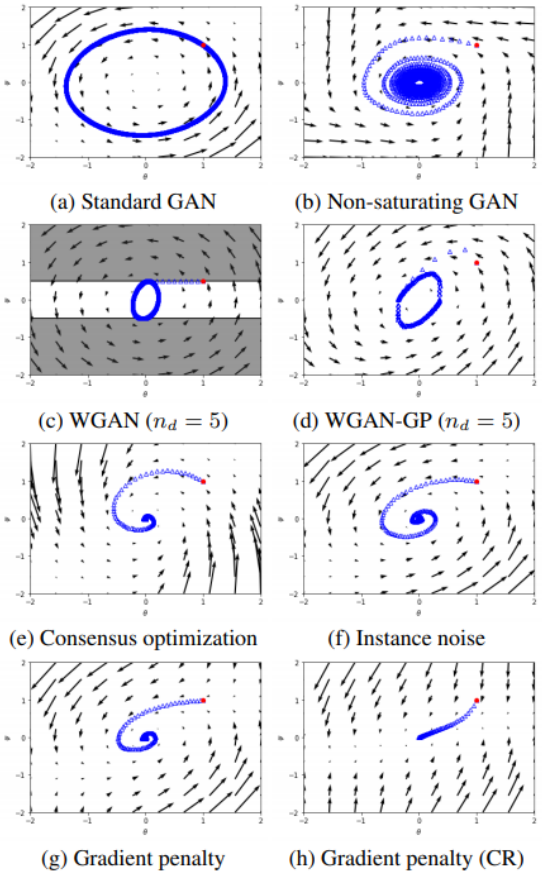
Wasserstein GAN (WGAN) discriminator is constrained to be Lipschitz-continuous

Gradient penalty discriminator is constrained to be continuous even harder

Spectral norm discriminator is **really** constrained to be continuous

Instance noise add noise to the data and generated samples

these are pretty good choices today



Mescheder et al. **Which Training Methods for GANs do actually Converge**? 2108.

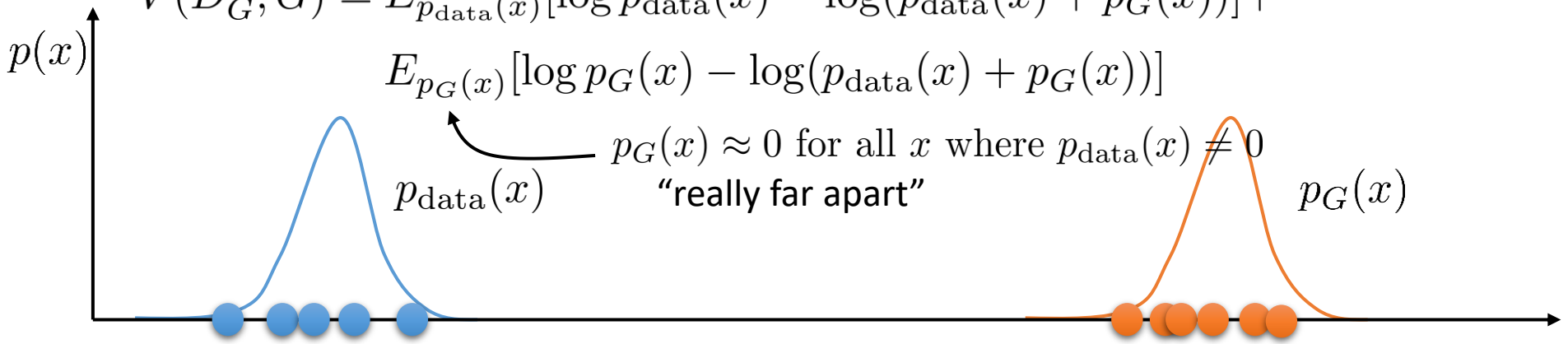
Wasserstein GAN (WGAN)

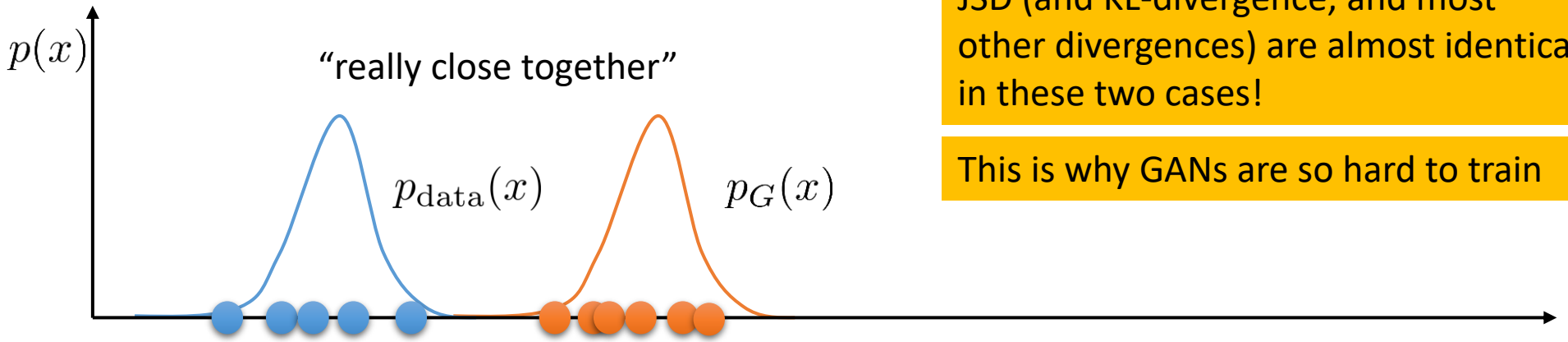
High-level intuition: the JS divergence used by the classic GAN doesn't account for "distance"

$p_{\text{data}}(x) \approx 0$ for all x where $p_G(x) \neq 0$

$$V(D_G^\star, G) = E_{p_{\text{data}}(x)}[\log p_{\text{data}}(x) - \log(p_{\text{data}}(x) + p_G(x))] +$$
$$E_{p_G(x)}[\log p_G(x) - \log(p_{\text{data}}(x) + p_G(x))]$$

$p_G(x) \approx 0$ for all x where $p_{\text{data}}(x) \neq 0$





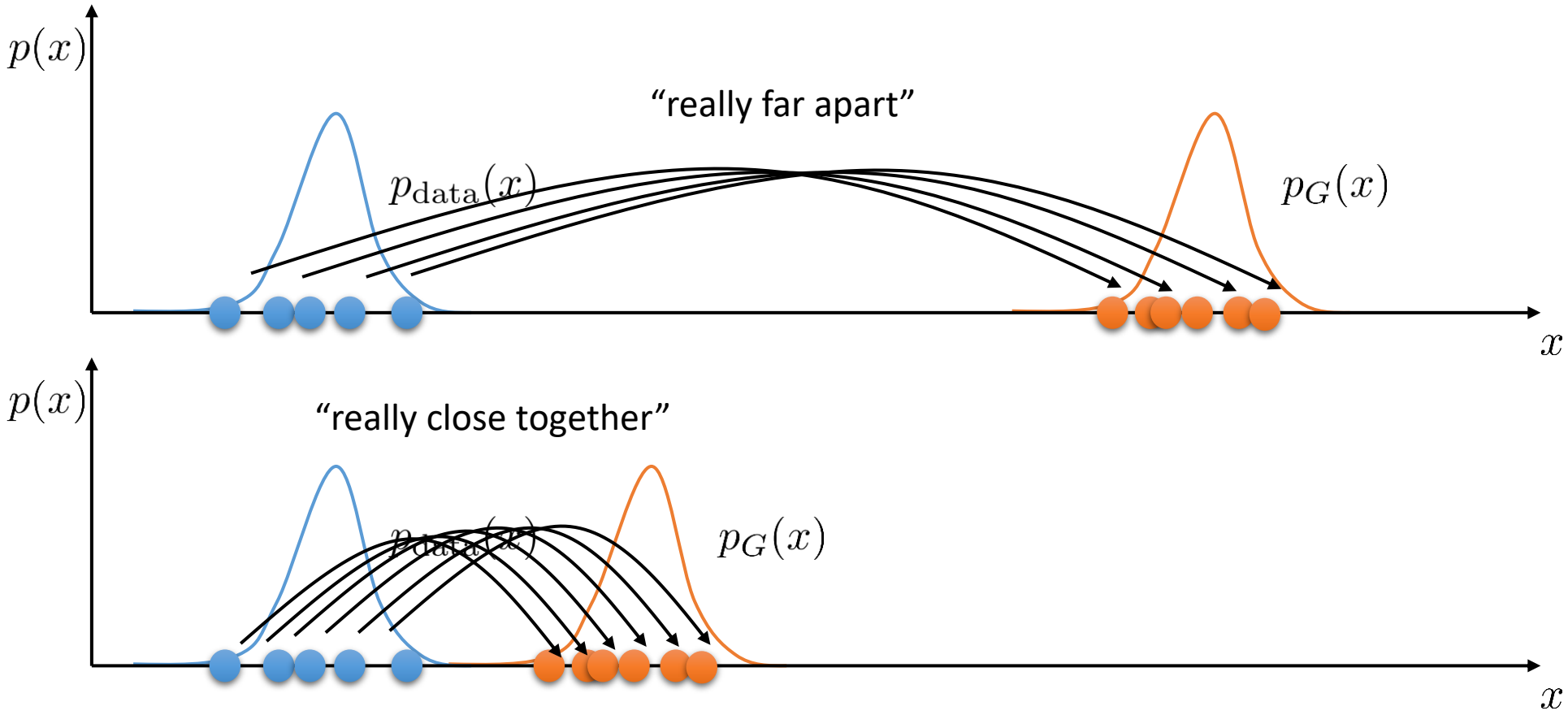
JSD (and KL-divergence, and most other divergences) are almost identical in these two cases!

This is why GANs are so hard to train

Wasserstein GAN (WGAN)

A better metric: consider how far apart (in Euclidean space) all the "bits" of probability are

More precisely: optimal transport ("Earth mover's distance") – how far do you have to go to "transport" one distribution into another



Wasserstein GAN (WGAN)

A better metric: consider how far apart (in Euclidean space) all the "bits" of probability are

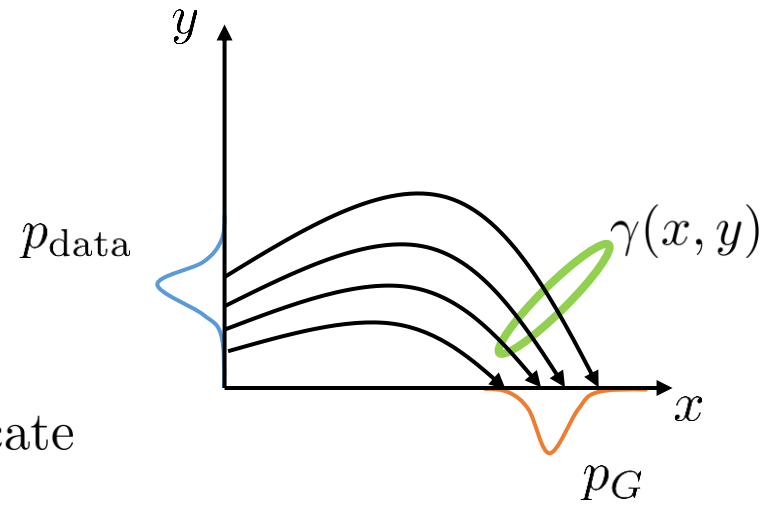
More precisely: optimal transport ("Earth mover's distance") – how far do you have to go to "transport" one distribution into another

Formal definition: (don't worry too much if this is hard to understand, not actually necessary to *implement* WGAN)

$$W(p_{\text{data}}, p_G) = \inf_{\gamma} E_{(x,y)\sim\gamma(x,y)}[||x - y||]$$

$\gamma(x, y)$ is a *distribution* over x, y
with marginals $\gamma_X(x) = p_{\text{data}}(x)$ and $\gamma_Y(x) = p_G(x)$

intuition: correlations between x and y in $\gamma(x, y)$ indicate which x should be "transported" to which y



Wasserstein GAN (WGAN)

A better metric: consider how far apart (in Euclidean space) all the "bits" of probability are

More precisely: optimal transport ("Earth mover's distance") – how far do you have to go to "transport" one distribution into another

$$W(p_{\text{data}}, p_G) = \inf_{\gamma} E_{(x,y)\sim\gamma(x,y)}[||x - y||]$$

could learn γ directly but this is very hard

$p_{\text{data}}(x)$ is unknown

$\gamma(x, y)$ is really complex

cool theorem based on Kantorovich-Rubinstein duality:

(won't prove here, but uses tools from duality, similar to what you might learn when you study Lagrangian duality in a class on convex optimization)

$$W(p_{\text{data}}, p_G) = \sup_{||f||_L \leq 1} E_{p_{\text{data}}}[f(x)] - E_{p_G(x)}[f(x)]$$

set of all 1-Lipschitz scalar functions

$$|f(x) - f(y)| \leq |x - y|$$

equivalent to saying function has bounded slope
i.e., it should not be too steep

expressed as difference of expectations under $p_G(x)$ and $p_{\text{data}}(x)$ just like a regular GAN!

How?

Wasserstein GAN (WGAN)

A better metric: consider how far apart (in Euclidean space) all the "bits" of probability are

More precisely: optimal transport ("Earth mover's distance") – how far do you have to go to "transport" one distribution into another

$$W(p_{\text{data}}, p_G) = \sup_{||f||_L \leq 1} E_{p_{\text{data}}}[f(x)] - E_{p_G(x)}[f(x)]$$

set of all 1-Lipschitz scalar functions

$$|f(x) - f(y)| \leq |x - y|$$

doesn't guarantee 1-Lipschitz unless we pick bounds very carefully

does guarantee K-Lipschitz for some finite K

idea: if f is a neural net with ReLU activations, can bound the weights W

1-layer: $f_\theta(x) = \text{ReLU}(W_1 x + b_1) \quad \theta = \{W_1, b_1\}$

if $W_{1,i,j} \in [-0.01, 0.01]$, slope can't be bigger than $0.01 \times D$

2-layer: $f_\theta(x) = W_2 \text{ReLU}(W_1 x + b_1) + b_2 \qquad \theta = \{W_1, b_1, W_2, b_2\}$

if $W_{\ell,i,j} \in [-0.01, 0.01]$, slope is bounded (but greater than 0.01!)

Wasserstein GAN (WGAN)

A better metric: consider how far apart (in Euclidean space) all the "bits" of probability are

More precisely: optimal transport ("Earth mover's distance") – how far do you have to go to "transport" one distribution into another

$$W(p_{\text{data}}, p_G) = \sup_{||f||_L \leq 1} E_{p_{\text{data}}}[f(x)] - E_{p_G(x)}[f(x)]$$

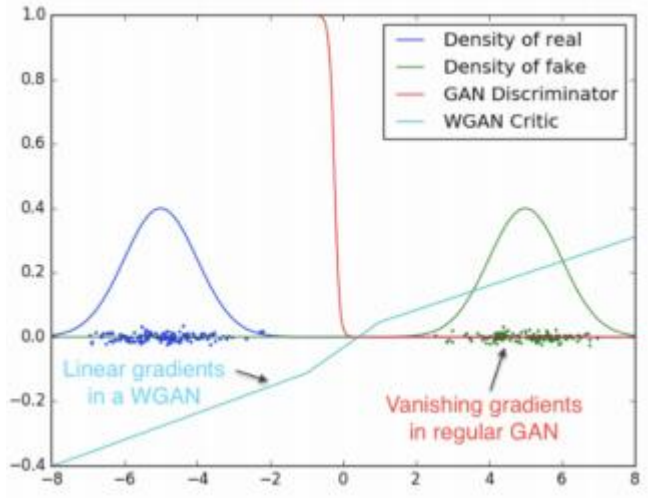
> Weight clipping is a clearly terrible way to enforce a Lipschitz constraint. If the clipping parameter is large, then it can take a long time for any weights to reach their limit, thereby making it harder to train the critic till optimality. If the clipping is small, this can easily lead to vanishing gradients when the number of layers is big, or batch normalization is not used (such as in RNNs). We experimented with simple variants (such as projecting the weights to a sphere) with little difference, and we stuck with weight clipping due to its simplicity and already good performance. However, we do leave the topic of enforcing Lipschitz constraints in a neural network setting for further investigation, and we actively encourage interested researchers to improve on this method.

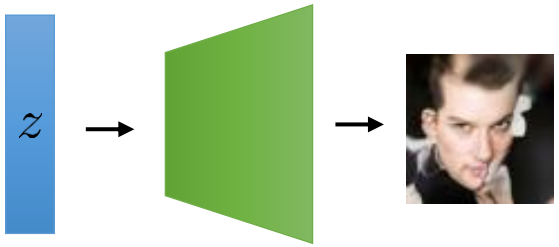
Arjovsky et al. Wasserstein GAN. 2017.

Wasserstein GAN (WGAN)

$$W(p_{\text{data}}, p_G) = \sup_{||f||_L \leq 1} E_{p_{\text{data}}}[f(x)] - E_{p_G(x)}[f(x)]$$

1. update f_θ using gradient of $E_{x \sim p_{\text{data}}}[f_\theta(x)] - E_{z \sim p(z)}[f_\theta(G(x))]$
2. clip all weight matrices in θ to $[-c, c]$
3. update generator to *maximize* $E_{z \sim p(z)}[f_\theta(G(z))]$





$p(z)$ $G(z)$

"generator"

real number

"discriminator"

discriminator outputs real numbers (not true/false probability)

e.g., $f_\theta(x) = W_2 \text{ReLU}(W_1 x + b_1) + b_2$

discriminator uses *weight clipping*

Arjovsky et al. Wasserstein GAN. 2017.

Better discriminator regularization

Weight clipping is a clearly terrible way to enforce a Lipschitz constraint.

Gradient penalty: Want bounded slope? We'll give you bounded slope!

1-Lipschitz:
$|f(x) - f(y)| \leq |x - y|$

update f_θ using gradient of

$$E_{x \sim p_{\text{data}}}[f_\theta(x) - \underbrace{\lambda(||\nabla_x f_\theta(x)||_2 - 1)^2}] - E_{z \sim p(z)}[f_\theta(G(z))]$$

make norm of gradient close to 1

for details, see: Gulrajani et al. **Improved Training of Wasserstein GANs**. 2017.

Spectral norm

Idea: bound the Lipschitz constant in terms of singular values of each W_ℓ

$f(x) = f_3 \circ f_2 \circ f_1(x)$ — neural net layers (e.g., linear, conv, ReLU, etc.)

$$||f(x)||_{\text{Lip}} = ||f_3 \circ f_2 \circ f_1||_{\text{Lip}} \leq ||f_3||_{\text{Lip}} \cdot ||f_2||_{\text{Lip}} \cdot ||f_1||_{\text{Lip}}$$

Lipschitz constant

to get intuition for why this is true, imagine these are **linear** functions

$\text{ReLU}(x) = \max(0, x) \Rightarrow$ max slope is 1!

that's easy, how about linear layers?

max slope of $Wx + b$ is **spectral norm**:

$$\sigma(W) = \max_{h:h\neq 0} \frac{||Wh||}{||h||} = \max_{||h||\leq 1} ||Wh||$$

largest singular value of W

Method: after each grad step, force $W_\ell \leftarrow \frac{W_\ell}{\sigma(W_\ell)}$

See paper for how to implement this efficiently

for details, see: Miyato et al. **Spectral Normalization for Generative Adversarial Networks**. 2018.

GAN training summary

- GAN training is really hard, because the discriminator can provide poor gradients

- Various “tricks” can make this much more practical
  - “Smooth” real-valued discriminators: LSGAN, WGAN, WGAN-GP, spectral norm
  - Instance noise

- The theory behind these tricks is quite complex, but the methods in practice are very simple

- Such GANs are **much** easier to train