Computational Systems Biology
Deep Learning in the Life Sciences

6.802 6.874 20.390 20.490 HST.506

David Gifford

Lecture 13

March 21, 2019

Generative Models

http://mit6874.github.io

What's on tap today!

- Generative models for synthesis
- Gradient methods
- Variational Autoencoders for synthesis
- Generative Adversarial Networks (GANs)
  - Standard formulation
  - Wasserstein loss

What you should know

- How gradient based optimization works
- Equational form for a variational autoencoder
- Equational form for a Generative Adversarial Network
- How a Wasserstein loss metric is defined

Why generative models?

Generative models perform synthesis

- Reveal what models understand
  - What is the best example of a written digit?
  - What is the best example of a celebrity?
- Transform examples with respect to one or more metrics
  - Improve sentiment of text
  - Perform multi-objective optimization of antibodies

- In generative modeling, we'd like to train a network that models a distribution, such as a distribution over images.
- One way to judge the quality of the model is to sample from it.
- This field has seen rapid progress:

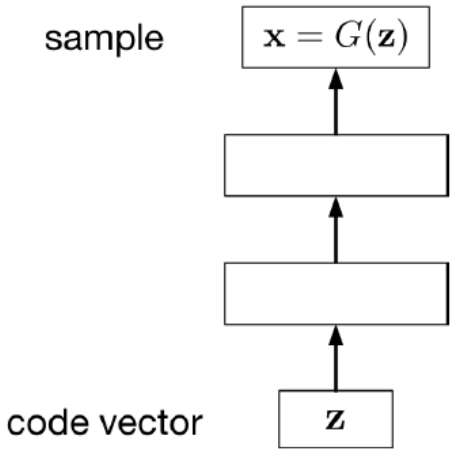
2009

2015

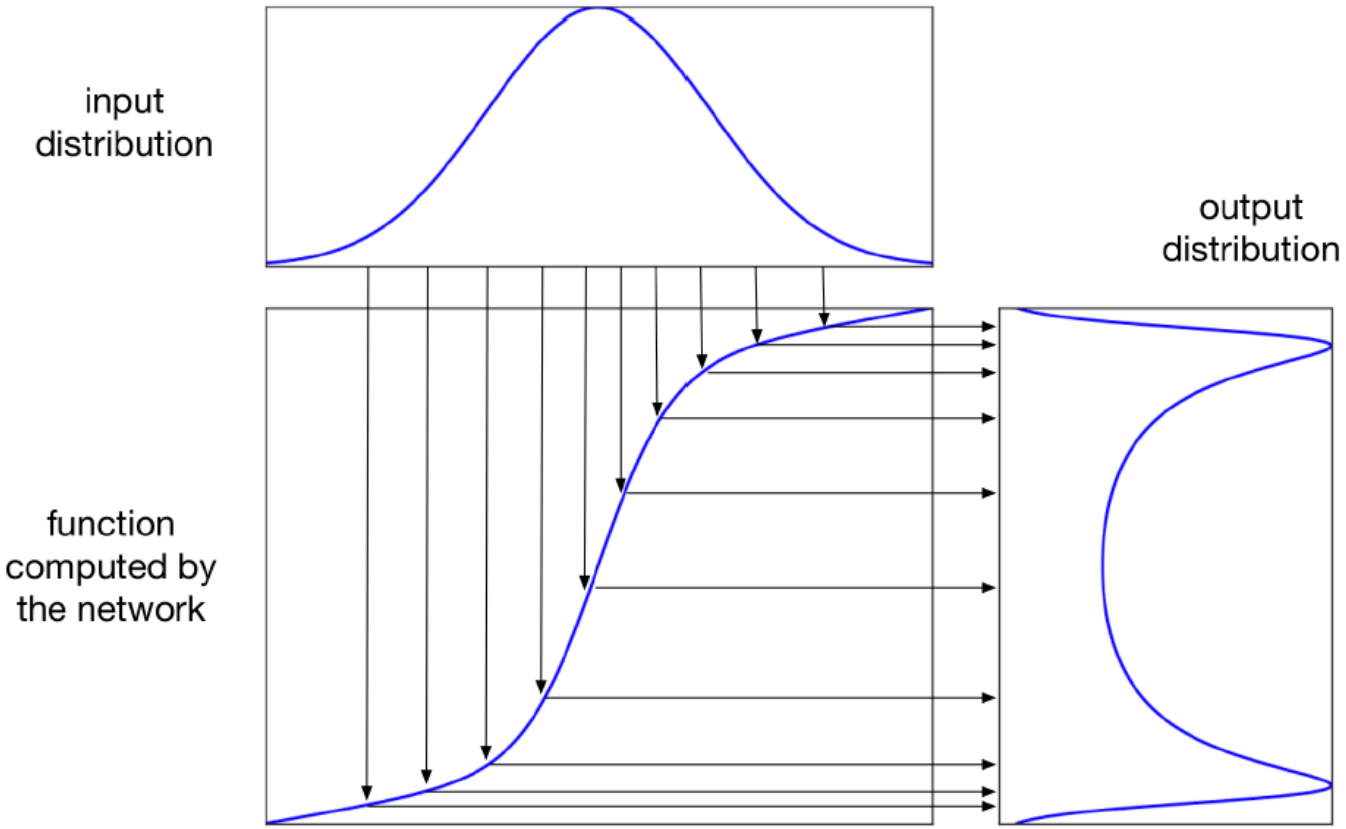
2018

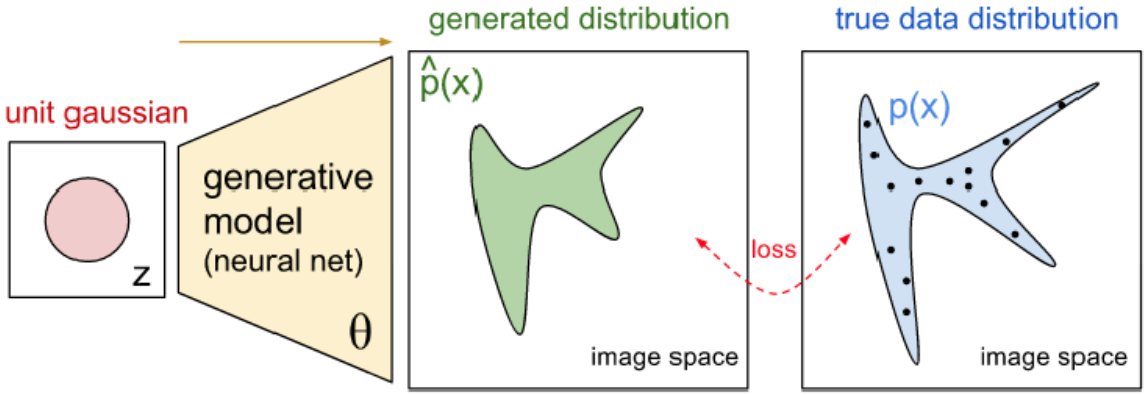
https://www.cs.toronto.edu/~rgrosse/courses/csc321_2018/slides/lec19.pdf

- Implicit generative models implicitly define a probability distribution
- Start by sampling the code vector $\mathbf{z}$ from a fixed, simple distribution (e.g. spherical Gaussian)
- The generator network computes a differentiable function G mapping $\mathbf{z}$ to an $\mathbf{x}$ in data space

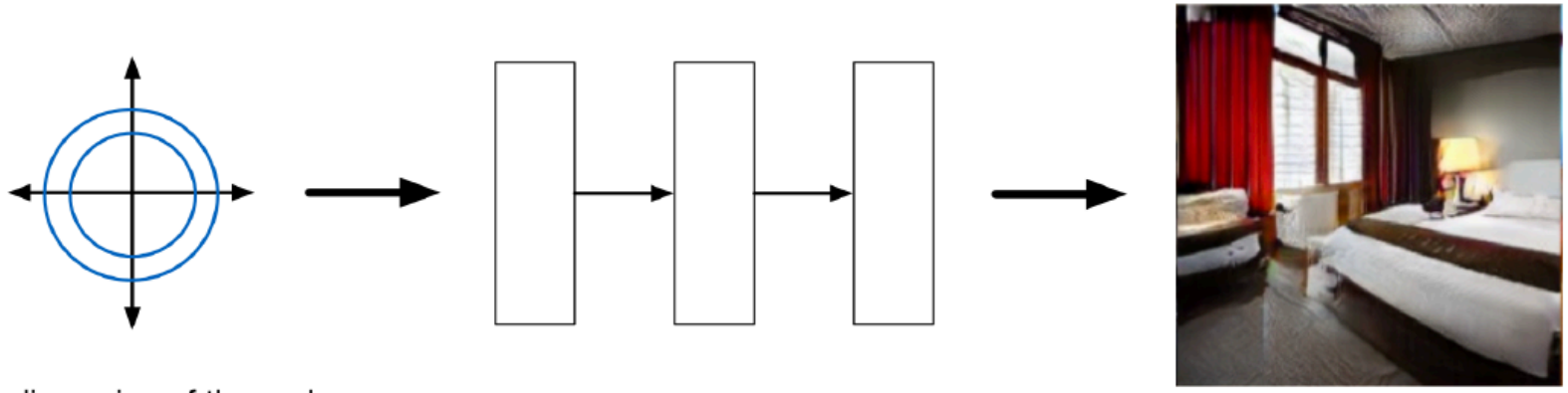


A 1-dimensional example:





https://blog.openai.com/generative-models/



Each dimension of the code vector is sampled independently from a simple distribution, e.g. Gaussian or uniform.

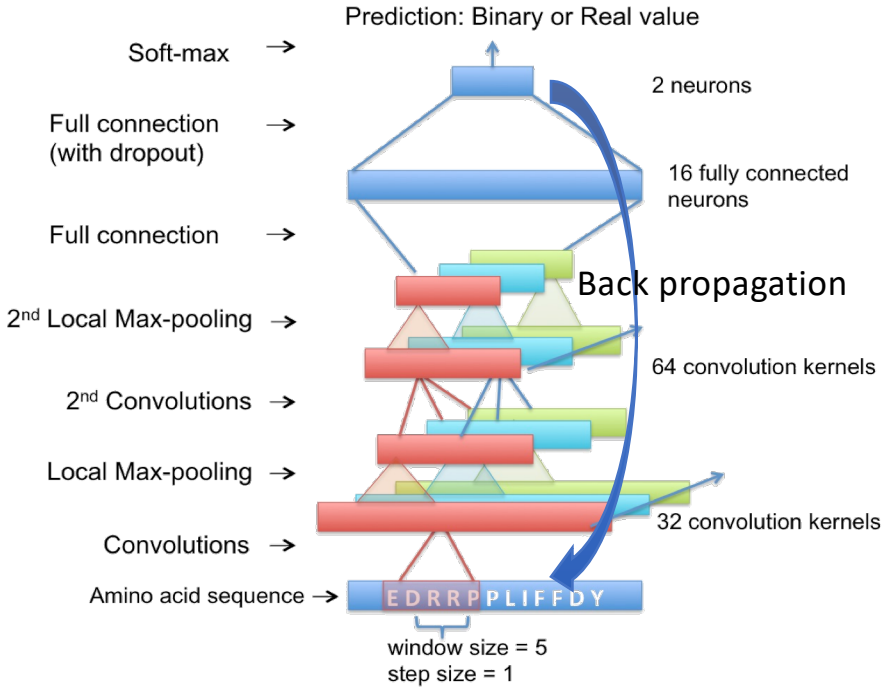
This is fed to a (deterministic) generator network.

The network outputs an image.

This sort of architecture sounded preposterous to many of us, but amazingly, it works.

Gradient based optimization

Use gradient to change input to minimize loss; model parameters fixed during earlier training



Issues with gradient based optimization

- Gradients may lead proposals outside of model training distribution
- Modeling may fail outside of training distribution leading to poor optimization
- Ensemble approaches and uncertainty metrics for optimized sequences can be used to eliminate optimized sequences with a low probability of improvement
- Note that complex objective functions can be used along with multi-output networks to perform multi-objective optimization

Variational Autoencoders can provide improved examples

Improving outcomes for structured sequences [12]

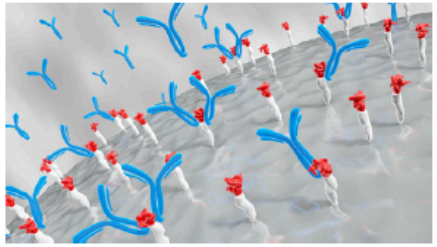
- Discrete sequence data is commonplace (eg. text, proteins/genes)

  sequence $x = (s_1, \ldots, s_T) \in \mathcal{X}$ where each symbol $s_t \in \mathcal{S}$ (discrete vocabulary)

- Tiny fraction of $\mathcal{X}$ represents sequences likely to occur naturally

  (ie. those which appear *realistic*)

- Each sequence x is associated with outcome $y \in \mathbb{R}$



$\hookrightarrow y, \quad x =$ ASVKVSKC

[12] J. Mueller et al. Sequence to better sequence: Continuous revision of combinatorial structures. *ICML*, 2017.

Problem setup

- Dataset $\mathcal{D}_n = \{(x_i, y_i)\}_{i=1}^n \overset{iid}{\sim} p_{XY}$ of sequence-outcome pairs
- $p_X =$ generative model of the *natural* sequences (unknown)

Goal: Given new sequence $x_0 \sim p_X$ (with unknown outcome), quickly identify a revision x^* with superior expected outcome

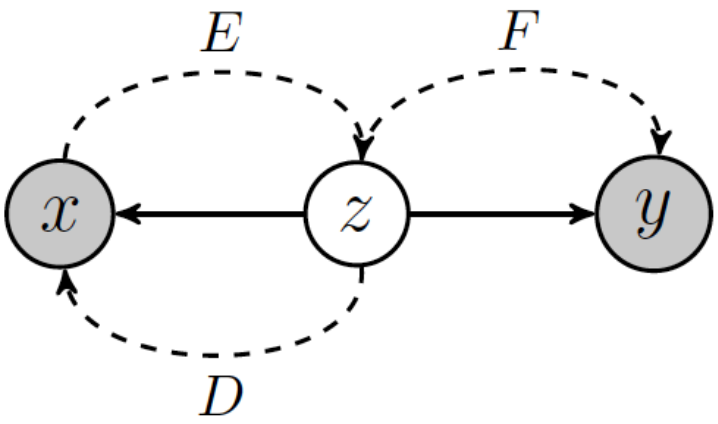
$$x^* = \underset{x \in \mathcal{C}_{x_0}}{\operatorname{argmax}} \ \mathbb{E}[Y \mid X = x]$$

$\mathcal{C}_{x_0} \subset \mathcal{X} =$ feasible set of sequences which appear natural and are minor revisions of x_0

Desiderata for our revision procedure: $x_0 \rightarrow x^*$

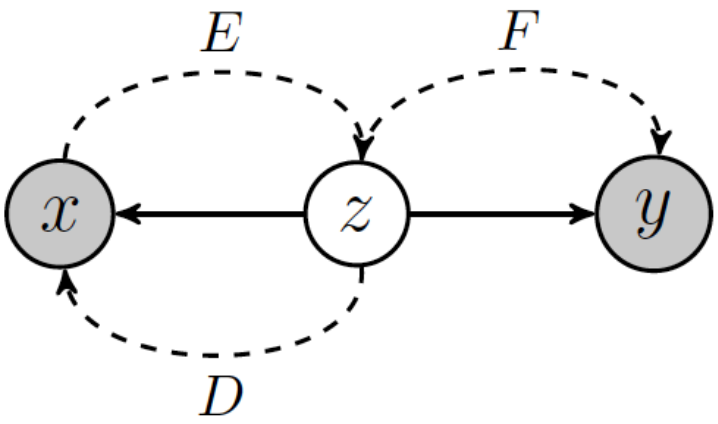
- Produces natural sequences
- Preserves intrinsic similarity
- Improves outcomes
- Computationally efficient

Probabilistic generative model



- Continuous latent factors $Z \in \mathbb{R}^d$ produce sequence X + outcome Y
  - Prior: $p_Z = N(0, \mathbf{I})$
- Approximate inference maps $F \simeq y|z$, $E \simeq z|x$ & likelihood function $D \simeq x|z$ parameterized via three neural networks $\mathcal{F}, \mathcal{E}, \mathcal{D}$
- Parameters of $\mathcal{F}, \mathcal{E}, \mathcal{D}$ learned jointly using various objective functions that are added together

Probabilistic generative model



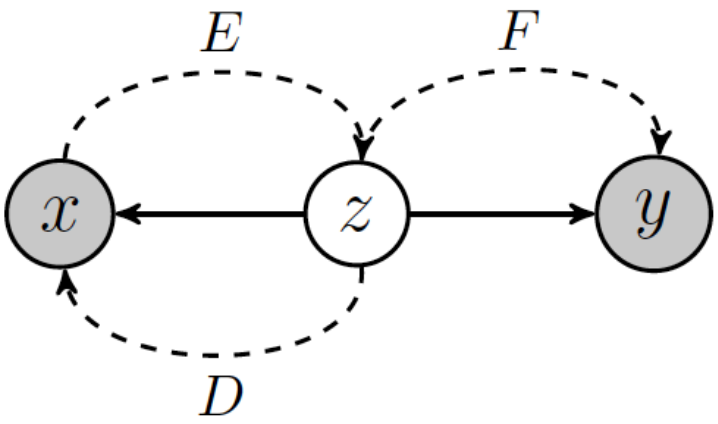
$$p_Z = N(0, \mathbf{I})$$

Why is this important?

Why does it make the task difficult?

- Continuous latent factors $Z \in \mathbb{R}^d$ produce sequence X + outcome Y
  - Prior: $p_Z = N(0, \mathbf{I})$
- Approximate inference maps $F \simeq y|z$, $E \simeq z|x$ & likelihood function $D \simeq x|z$ parameterized via three neural networks $\mathcal{F}, \mathcal{E}, \mathcal{D}$
- Parameters of $\mathcal{F}, \mathcal{E}, \mathcal{D}$ learned jointly using various objective functions that are added together

Probabilistic generative model



$$p_Z = N(0, \mathbf{I})$$

Why is this important?

Find plausible revisions

Why does it make the task difficult?

p(z | x) intractable

- Continuous latent factors $Z \in \mathbb{R}^d$ produce sequence X + outcome Y
  - Prior: $p_Z = N(0, \mathbf{I})$
- Approximate inference maps $F \simeq y|z$, $E \simeq z|x$ & likelihood function $D \simeq x|z$ parameterized via three neural networks $\mathcal{F}, \mathcal{E}, \mathcal{D}$
- Parameters of $\mathcal{F}, \mathcal{E}, \mathcal{D}$ learned jointly using various objective functions that are added together

Variational autoencoder (VAE)[13]

- Generative model for sequences: $z \sim p_Z, \; x \sim \underbrace{p_D(x \mid z)}_{\text{parameterized by RNN } \mathcal{D}}$ $\qquad p_Z = N(0, \mathbf{I})$
- Variational posterior approximation:

$$p(z \mid x) \approx \underbrace{N(\mu_{z|x}, \text{diag}(\sigma^2_{z|x}))}_{q_E(z \mid x) \text{ parameterized by RNN } \mathcal{E}}$$

- Learn parameters of $\mathcal{E}, \mathcal{D}$ using stochastic variational inference to maximize: $\mathbb{E}_{q_E(z|x)}[\log p_D(x \mid z)] - \text{KL}(q_E(z \mid x) || \, p_Z) \leqslant \log p_X(x)$
- $E(x) := \underset{z \in \mathbb{R}^d}{\text{argmax}} \; q_E(z \mid x) = \mu_{z|x}$ (most likely Z that generated sequence x)
- $D(z) := \underset{x \in \mathcal{X}}{\text{argmax}} \; p_D(x \mid z)$ (most likely sequence generated from $Z = z$)

[13] D. Kingma and M. Welling. Auto-encoding variational Bayes. *ICLR*, 2014.

Compositional prediction of outcomes

- Outcome map: $\underbrace{F(z)}_{\text{parameterized by feedforward network } \mathcal{F}} = \mathbb{E}[Y \mid Z = z]$
- Employ Taylor approximation: $F(E(x)) \approx \mathbb{E}[Y \mid X = x]$
- Jointly train $\mathcal{E}$, $\mathcal{F}$ to minimize mean squared error $= [y - F(E(x))]^2$

Example of a bad latent representation

Suppose that for $x \in \mathcal{X}$:

$$E(x) = z = \begin{bmatrix} z_1 \\ z_2 \end{bmatrix} \in \mathbb{R}^d$$

$$\widehat{y} = F(z) = F(z_1) \qquad \text{and} \qquad \widehat{x} = D(z) = D(z_2)$$

Outcome prediction ignores z_2 Decoding step ignores z_1

- Changing $z_1 \implies$ change in predicted outcome $\widehat{y}$
  but *no* change in resulting sequence $\widehat{x}$ ☹

Enforcing invariance

- Can avoid bad scenario by bottlenecking latent dimensionality d
- Add invariance loss to overall training objective:

$$\mathbb{E}_{z \sim p_Z} \left[F(z) - F(E(D(z))) \right]^2$$

- $\mathcal{L}_{\text{inv}} \to 0$ ensures outcome-predictions remain invariant to encoding-decoding variation
- $\frac{\partial \mathcal{L}_{\text{inv}}}{\partial E}$, $\frac{\partial \mathcal{L}_{\text{inv}}}{\partial F}$ estimated via Monte-Carlo samples from p_Z

Overall VAE loss function

$$\mathcal{L}(x, y) = \mathcal{L}_{\text{rec}} + \lambda_{\text{pri}}\mathcal{L}_{\text{pri}} + \frac{\lambda_{\text{mse}}}{\sigma_Y^2}\mathcal{L}_{\text{mse}} + \frac{\lambda_{\text{inv}}}{\sigma_Y^2}\mathcal{L}_{\text{inv}}$$

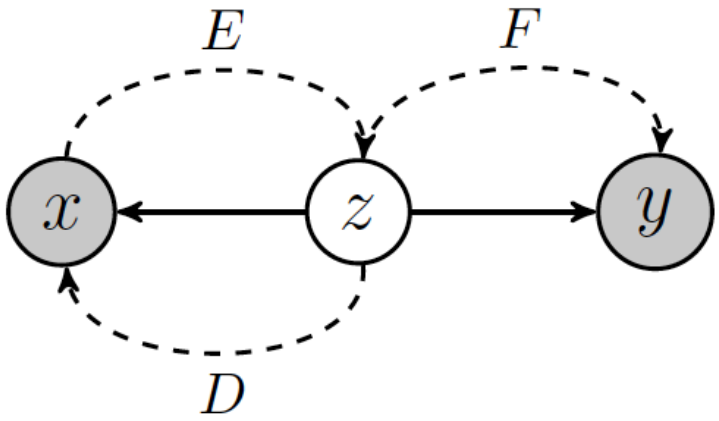
$$\log p_X(x) \geqslant -\left[\mathcal{L}_{\text{rec}}(x) + \mathcal{L}_{\text{pri}}(x)\right]$$

$$\mathcal{L}_{\text{rec}}(x) = -\mathbb{E}_{q_E(z|x)}\left[\log p_D(x \mid z)\right]$$

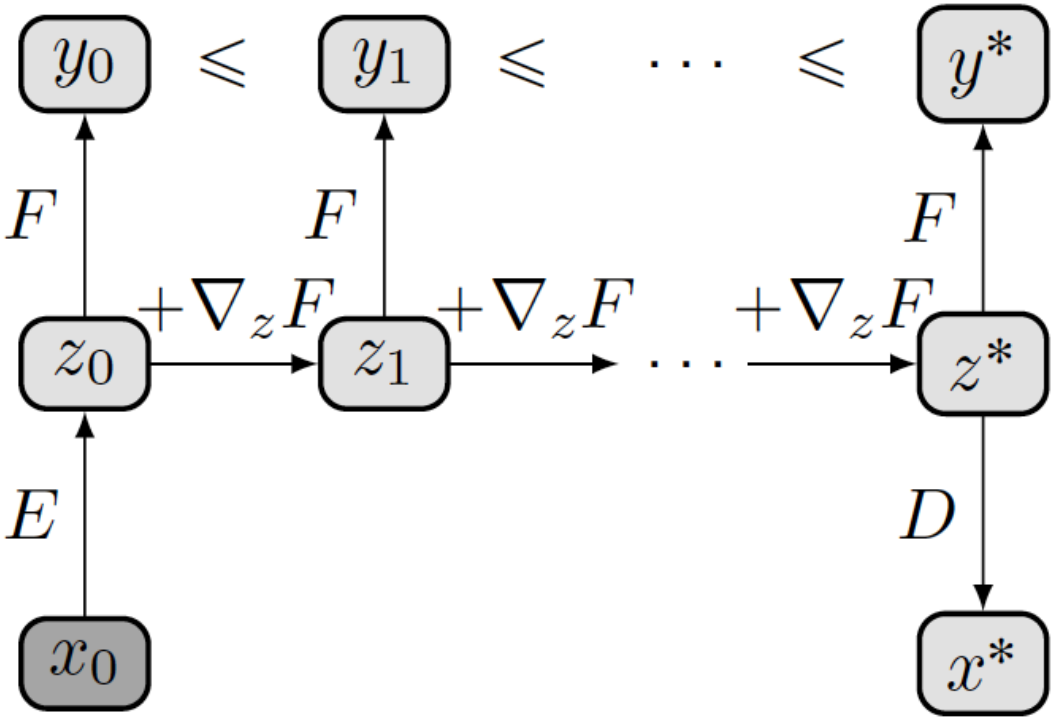
$$\mathcal{L}_{\text{pri}}(x) = \mathbf{KL}(q_E(z \mid x) || \, p_Z)$$

$$\mathcal{L}_{\text{mse}}(x, y) = \left[y - F(E(x))\right]^2$$

$$\mathcal{L}_{\text{inv}} = \mathbb{E}_{z \sim p_Z}\left[F(z) - F(E(D(z)))\right]^2$$



Revision framework



Proposing revisions

REVISE Algorithm

Input: sequence $x_0 \in \mathcal{X}$, constant $\alpha \in (0, |2\pi\Sigma_{z|x_0}|^{-\frac{1}{2}})$

Output: revised sequence $x^* \in \mathcal{X}$

1) Use $\mathcal{E}$ to compute $q_E(z \mid x_0),\ E(x_0) = \mathbb{E}_{q_E}[z \mid x_0]$

2) Define $\mathcal{C}_{x_0} = \{z \in \mathbb{R}^d : q_E(z \mid x_0) \geqslant \alpha\}$ (ellipsoid)

3) Find $z^* \approx \underset{z \in \mathcal{C}_{x_0}}{\operatorname{argmax}}\ F(z)$ (gradient ascent w/ log-barrier penalty)

4) Return $x^* = D(z^*) \approx \underset{x \in \mathcal{X}}{\operatorname{argmax}}\, p_D(x \mid z^*)$ (greedy beam search)

Improving sentence positivity

- Data = 1M+ short sentences from BeerAdvocate reviews
- $y \in [0, 1]$: VADER sentiment compound score of each sentence[14]
- Apply methods to revise set of 1000 held-out sentences

[14] C. Hutto and E. Gilbert. Vader: A parsimonious rule-based model for sentiment analysis of social media text. *ICWSM*, 2014.

Improving sentence positivity

Model	$\Delta_Y(x^*)$	$L(x^*)$	$d(x^*, x_0)$
$\log \alpha = -10000$	**0.52** ±0.77	-8.8 ±6.5	2.6 ±3.3
$\log \alpha = -1$	0.31 ±0.50	**-7.6** ±5.8	1.7 ±2.6
$\lambda_{\text{inv}} = \lambda_{\text{pri}} = 0$	0.22 ±1.03	-10.2 ±7.0	3.3 ±3.4
SEARCH	0.19 ±0.56	-7.7 ±4.2	3.0 ±1.2

$\Delta_Y(x^*)$ = outcome improvement from revision (rescaled by variance of outcomes)

$L(x^*) = \hat{p}_X(x^*) - \hat{p}_X(x_0)$, $\hat{p}_X$ = likelihood estimated via language model

$d(x^*, x_0)$ = Levenshtein (edit) distance

Improving sentence positivity

Model	Sentence	$\Delta_Y(x^*)$	$\Delta_L(x^*)$
x_0	**this smells pretty bad.**	-	-
$\log \alpha = -10000$	smells pretty delightful!	+2.8	-0.5
$\log \alpha = -1$	i liked this smells pretty.	+2.5	-2.8
$\lambda_{\text{inv}} = \lambda_{\text{pri}} = 0$	pretty this smells bad!	-0.2	-3.1
SEARCH	wow this smells pretty bad.	+1.9	-4.6
x_0	**i like to support san diego beers.**	-	-
$\log \alpha = -10000$	i love to support craft beers!	+0.5	+1.6
$\log \alpha = -1$	i like to support craft beers!	+0.1	+2.6
$\lambda_{\text{inv}} = \lambda_{\text{pri}} = 0$	i like to support you know.	0	+3.7
SEARCH	i like to super support san diego.	+0.7	-2.9
x_0	**i'm not sure how old the bottle is.**	-	-
$\log \alpha = -10000$	i definitely enjoy how old is the bottle is.	+3.0	-3.6
$\log \alpha = -1$	i'm sure not sure how old the bottle is.	+2.5	-6.8
$\lambda_{\text{inv}} = \lambda_{\text{pri}} = 0$	i'm sure better is the highlights when cheers.	+3.3	-9.2
SEARCH	i 'm not sure how the bottle is love.	+2.3	-3.3

Revising modern text in the language of Shakespeare

- Dataset of $\sim$100K short sentences
- Each is either from Shakespeare with label $y = 0.9$ or a more contemporary source (from NLTK) with label $y = 0.1$
- Given new sentence, revise so that author is increasingly expected to be Shakespeare rather than contemporary source

Revising modern text in the language of Shakespeare

# Steps	Decoded Sentence
x_0	**where are you, henry??**
100	where are you, henry??
1000	where are you, royal??
5000	where art thou now?
10000	which cannot come, you of thee?
x^*	where art thou, keeper??
x_0	**somewhere, somebody is bound to love us.**
100	somewhere, somebody is bound to love us.
1000	courage, honey, somebody is bound to love us!
5000	courage man; 'tis love that is lost to us.
10000	thou, within courage to brush and such us brush.
x^*	courage man; somebody is bound to love us.
x_0	**you are both the same size.**
100	you are both the same.
1000	you are both wretched.
5000	you are both the king.
10000	you are both these are very.
x^*	you are both wretched men.

Desiderata for our revision procedure

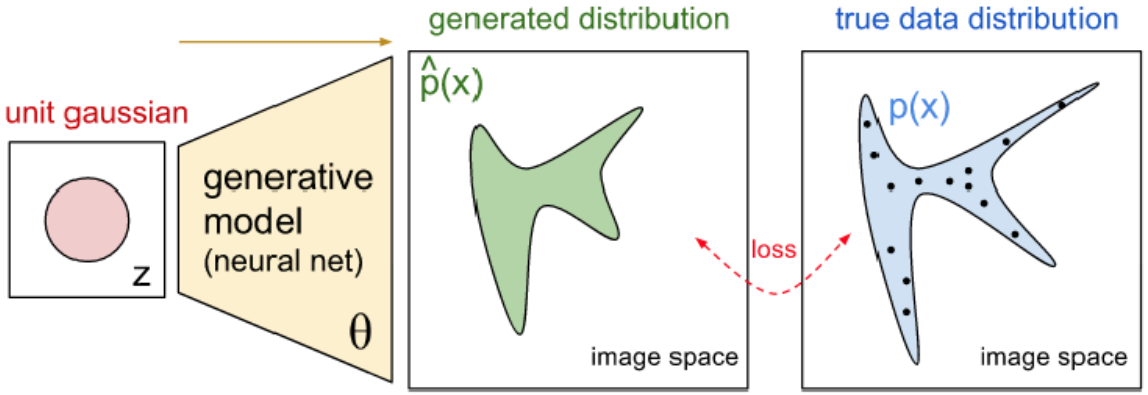
- Improves outcomes ✓
- Produces natural sequences ✓
- Computationally efficient ✓
- Preserves intrinsic similarity ✗

To model similarity, can leverage siamese RNN[15] that embeds sentences into vector space where L_1 metric reflects human-labeled similarity

[15] J. Mueller and A. Thyagarajan. Siamese Recurrent Architectures for Learning Sentence Similarity. *AAAI*, 2016.

Generative Adversarial Networks

We wish to learn a generative model that matches the true data distribution



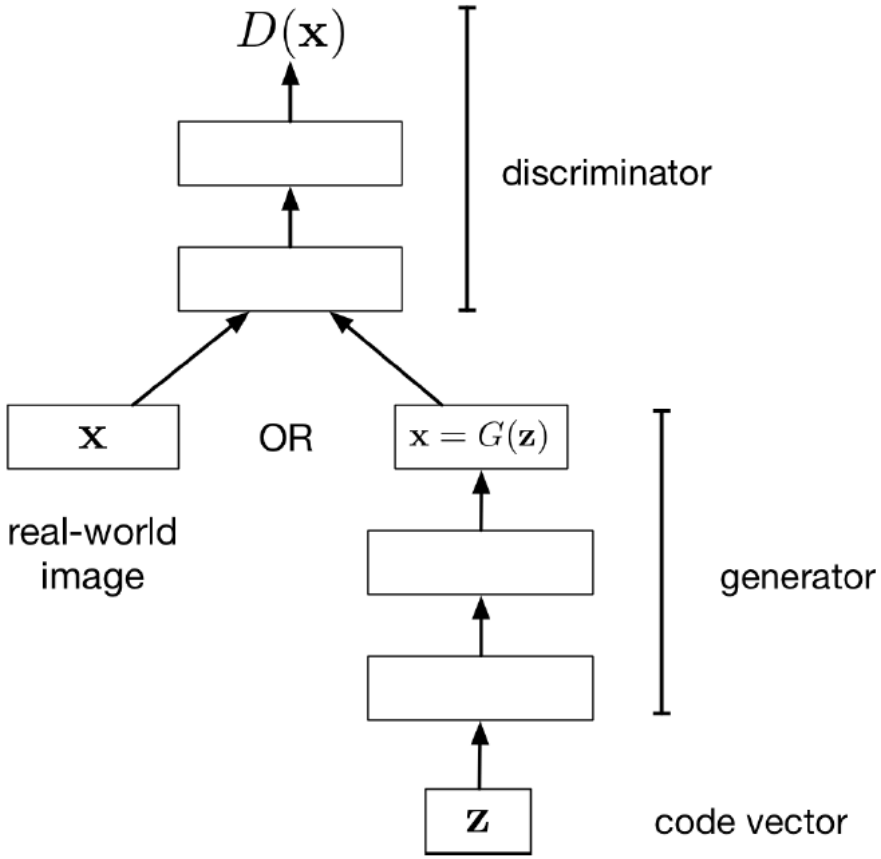
https://blog.openai.com/generative-models/

The Generative Adversarial Network (GAN) Game

- The advantage of implicit generative models: if you have some criterion for evaluating the quality of samples, then you can compute its gradient with respect to the network parameters, and update the network's parameters to make the sample a little better
- The idea behind Generative Adversarial Networks (GANs): train two different networks
  - The generator network tries to produce realistic-looking samples
  - The discriminator network tries to figure out whether an image came from the training set or the generator network
- The generator network tries to fool the discriminator network

https://www.cs.toronto.edu/~rgrosse/courses/csc321_2018/slides/lec19.pdf

D(x) is probability x is from the real-world



- Let D denote the discriminator's predicted probability of being data
- Discriminator's cost function: cross-entropy loss for task of classifying real vs. fake images

$$\mathcal{J}_D = \mathbb{E}_{\mathbf{x}\sim\mathcal{D}}[-\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z}}[-\log(1 - D(G(\mathbf{z})))]$$

- One possible cost function for the generator: the opposite of the discriminator's

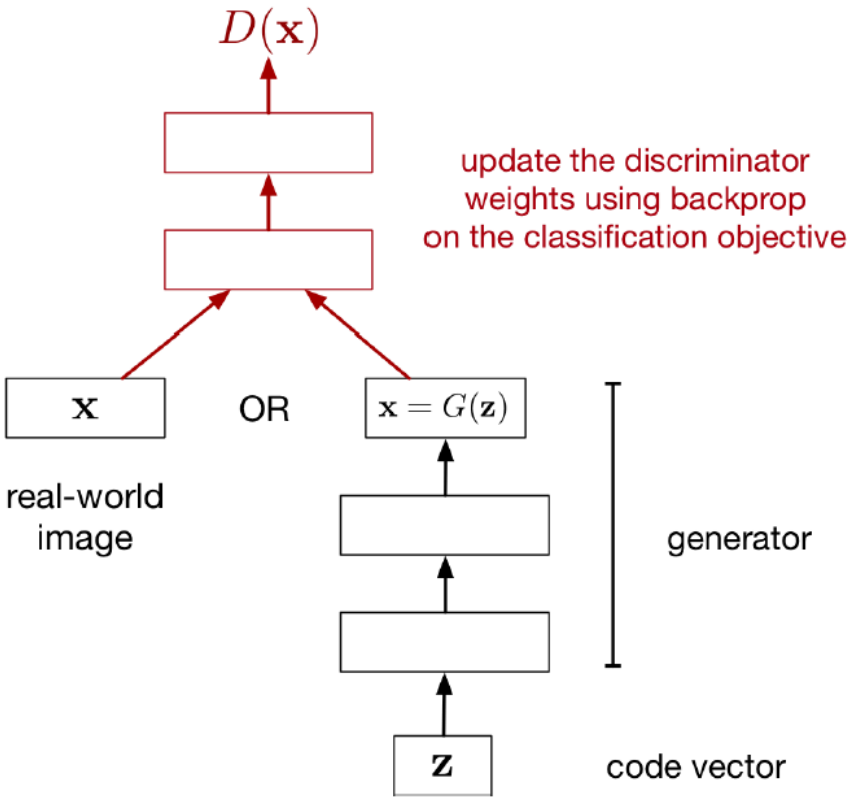
$$\begin{aligned}\mathcal{J}_G &= -\mathcal{J}_D \\ &= \text{const} + \mathbb{E}_{\mathbf{z}}[\log(1 - D(G(\mathbf{z})))]\end{aligned}$$

- This is called the minimax formulation, since the generator and discriminator are playing a zero-sum game against each other:

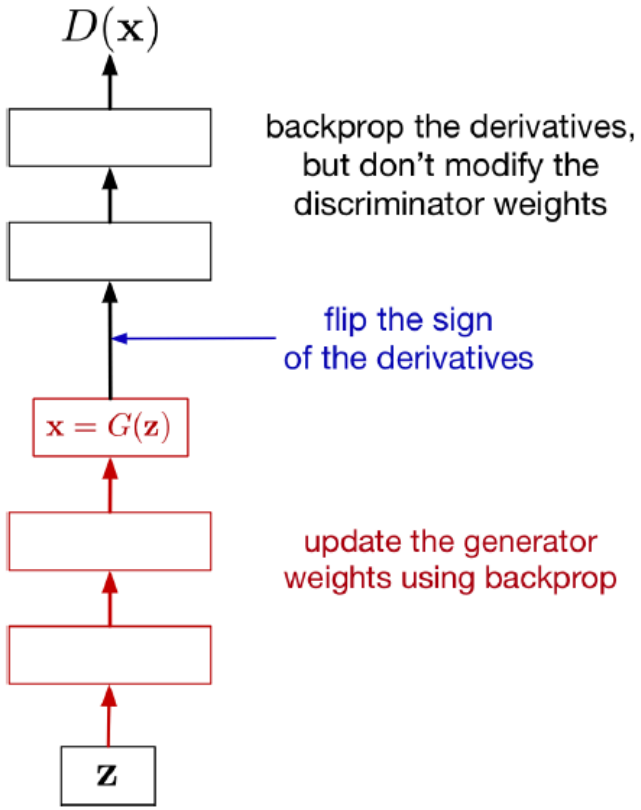
$$\max_G \min_D \mathcal{J}_D$$

Update discriminator to maximize D(real) and minimize D(G(z))

Updating the discriminator:



Update generator to maximize D(G(z))



We can improve the generator cost function

- We introduced the minimax cost function for the generator:

$$\mathcal{J}_G = \mathbb{E}_{\mathbf{z}}[\log(1 - D(G(\mathbf{z})))]$$

- One problem with this is saturation.
- Recall from our lecture on classification: when the prediction is really wrong,
  - "Logistic + squared error" gets a weak gradient signal
  - "Logistic + cross-entropy" gets a strong gradient signal
- Here, if the generated sample is really bad, the discriminator's prediction is close to 0, and the generator's cost is flat.

We can improve the generator cost function

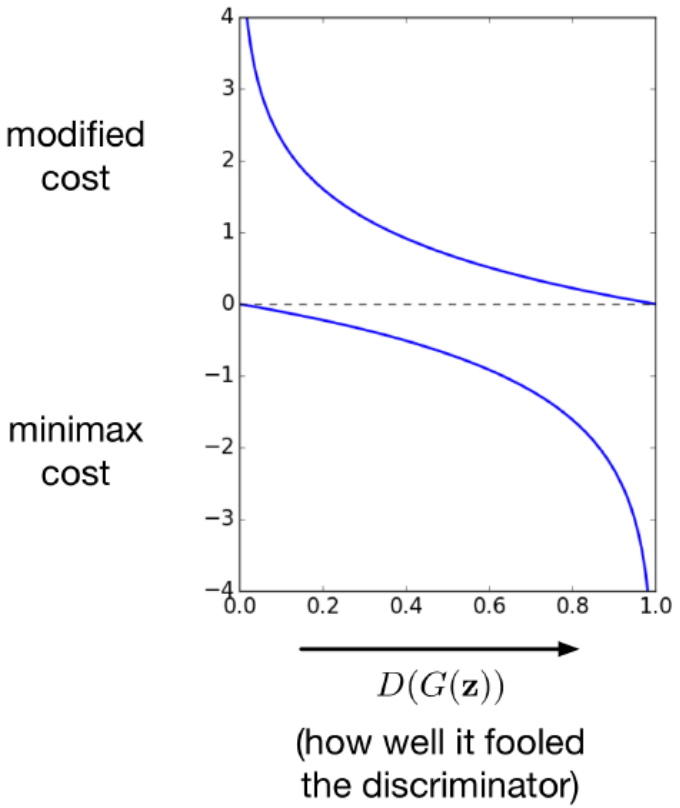
- Original minimax cost:

$$\mathcal{J}_G = \mathbb{E}_{\mathbf{z}}[\log(1 - D(G(\mathbf{z})))]$$

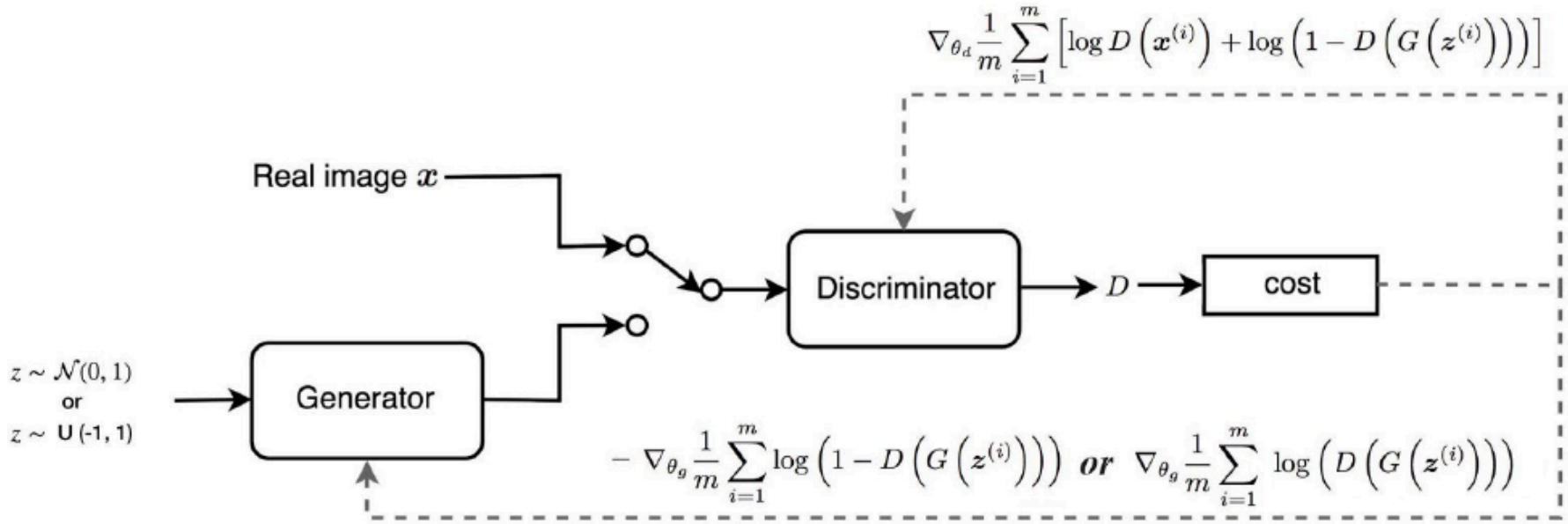
- Modified generator cost:

$$\mathcal{J}_G = \mathbb{E}_{\mathbf{z}}[-\log D(G(\mathbf{z}))]$$

- This fixes the saturation problem.



Summary of GAN objective functions



GANs have become a bit of a fad

- Since GANs were introduced in 2014, there have been hundreds of papers introducing various architectures and training methods.
- Most modern architectures are based on the Deep Convolutional GAN (DC-GAN), where the generator and discriminator are both conv nets.
- GAN Zoo: `https://github.com/hindupuravinash/the-gan-zoo`
  - Good source of horrible puns (VEEGAN, Checkhov GAN, etc.)

Celebrities:

Karras et al., 2017. Progressive growing of GANs for improved quality, stability, and variation

Bedrooms:

Karras et al., 2017. Progressive growing of GANs for improved quality, stability, and variation

Objects:

Karras et al., 2017. Progressive growing of GANs for improved quality, stability, and variation

GANs can fail

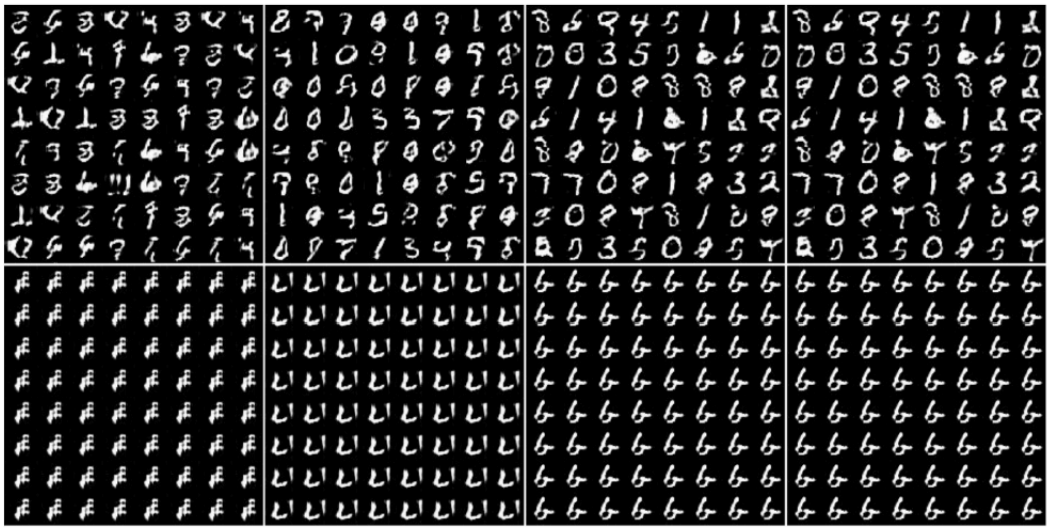
- GANs revolutionized generative modeling by producing crisp, high-resolution images.
- The catch: we don't know how well they're modeling the distribution.
  - Can't measure the log-likelihood they assign to held-out data.
  - Could they be memorizing training examples? (E.g., maybe they sometimes produce photos of real celebrities?)
  - We have no way to tell if they are dropping important modes from the distribution.
  - See Wu et al., "On the quantitative analysis of decoder-based generative models" for partial answers to these questions.

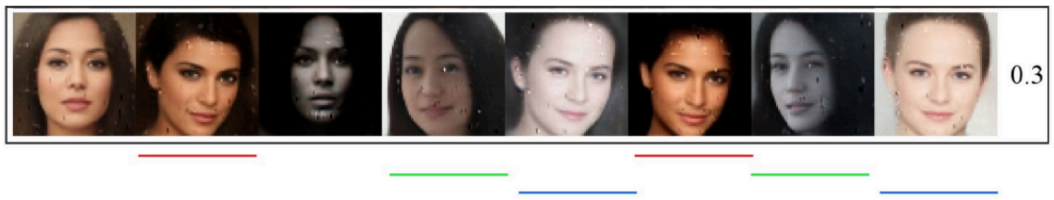
GAN Problems

- Non-convergence – model parameters oscillate and never converge
- Mode collapse – limited variety of samples from generator
- Diminished gradient – Discriminator is too successful and generator learns nothing
- Overfitting – imbalance between generator and discriminator
- Hyperparameter sensitivity – highly sensitive

https://medium.com/@jonathan_hui/gan-why-it-is-so-hard-to-train-generative-advisory-networks-819a86b3750b

Mode collapse





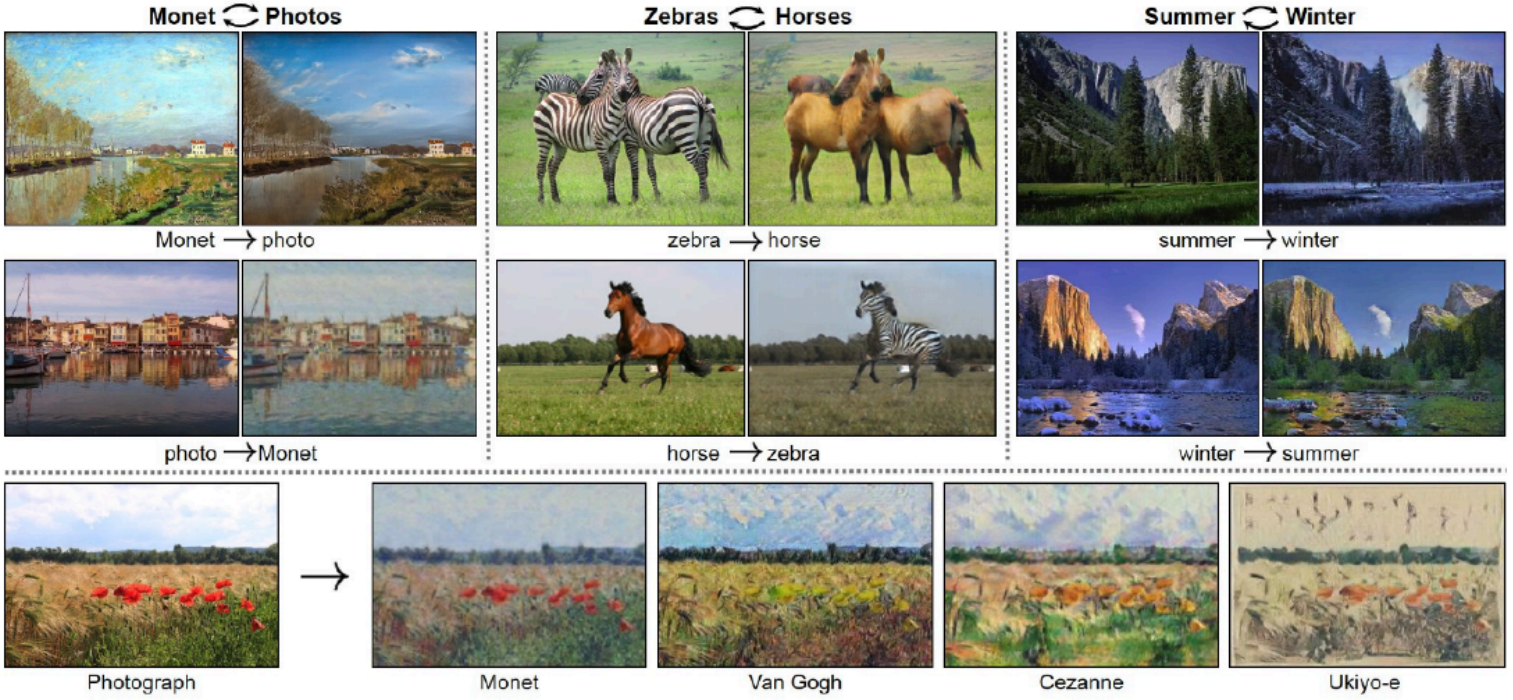
https://arxiv.org/pdf/1703.10717.pdf

CycleGANs map between styles

- If we had paired data (same content in both styles), this would be a supervised learning problem. But this is hard to find.
- The CycleGAN architecture learns to do it from unpaired data.
  - Train two different generator nets to go from style 1 to style 2, and vice versa.
  - Make sure the generated samples of style 2 are indistinguishable from real images by a discriminator net.
  - Make sure the generators are cycle-consistent: mapping from style 1 to style 2 and back again should give you almost the original image.

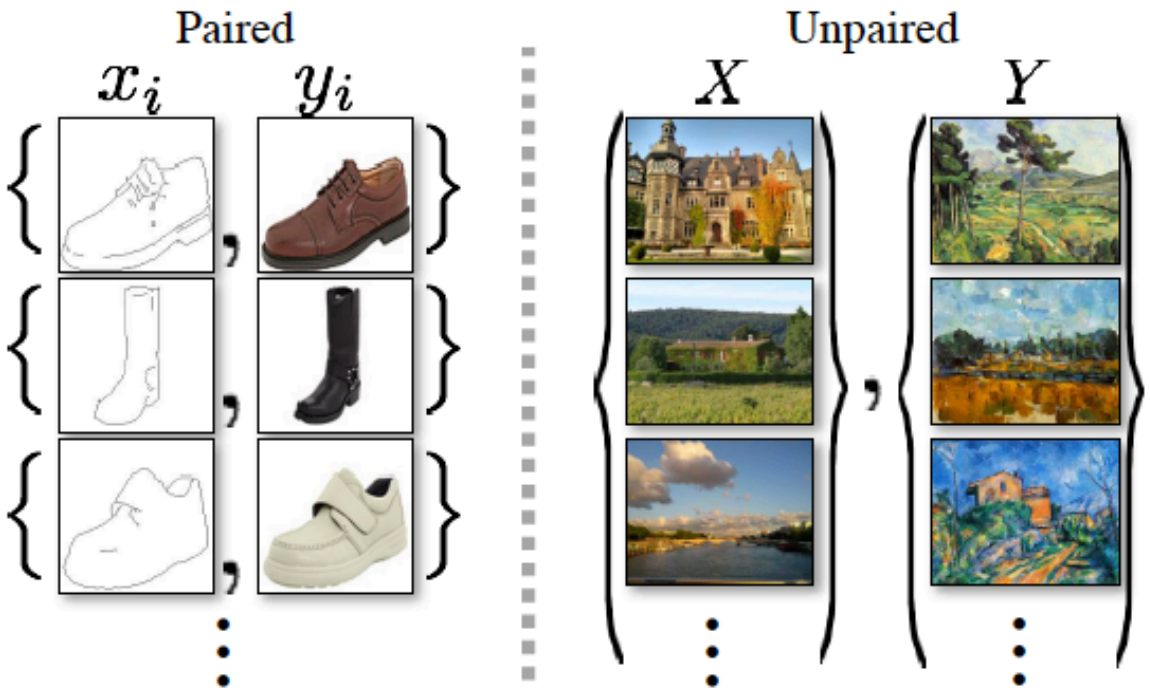
CycleGANs permit style transfer without matched training data

Style transfer problem: change the style of an image while preserving the content.



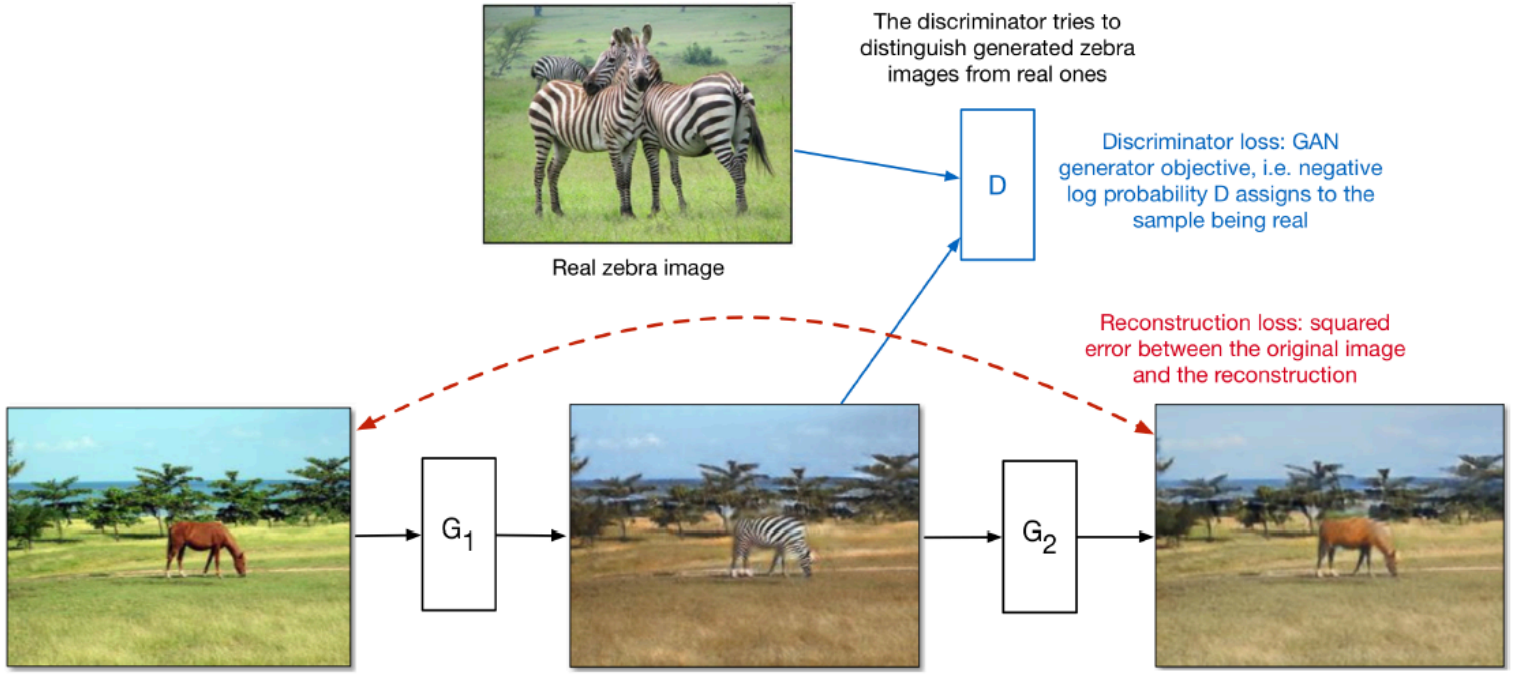
Data: Two unrelated collections of images, one for each style

CycleGANs permit style transfer without matched training data



https://arxiv.org/pdf/1703.10593.pdf

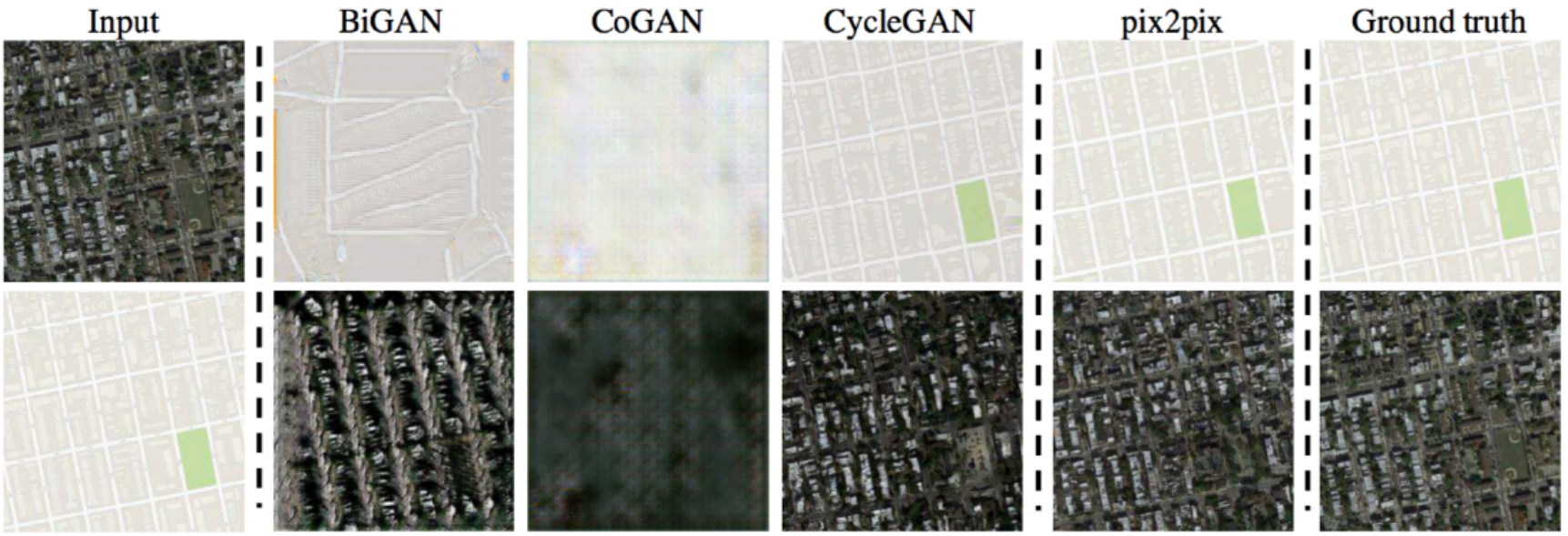
CycleGANs permit style transfer without matched training data



Total loss = discriminator loss + reconstruction loss

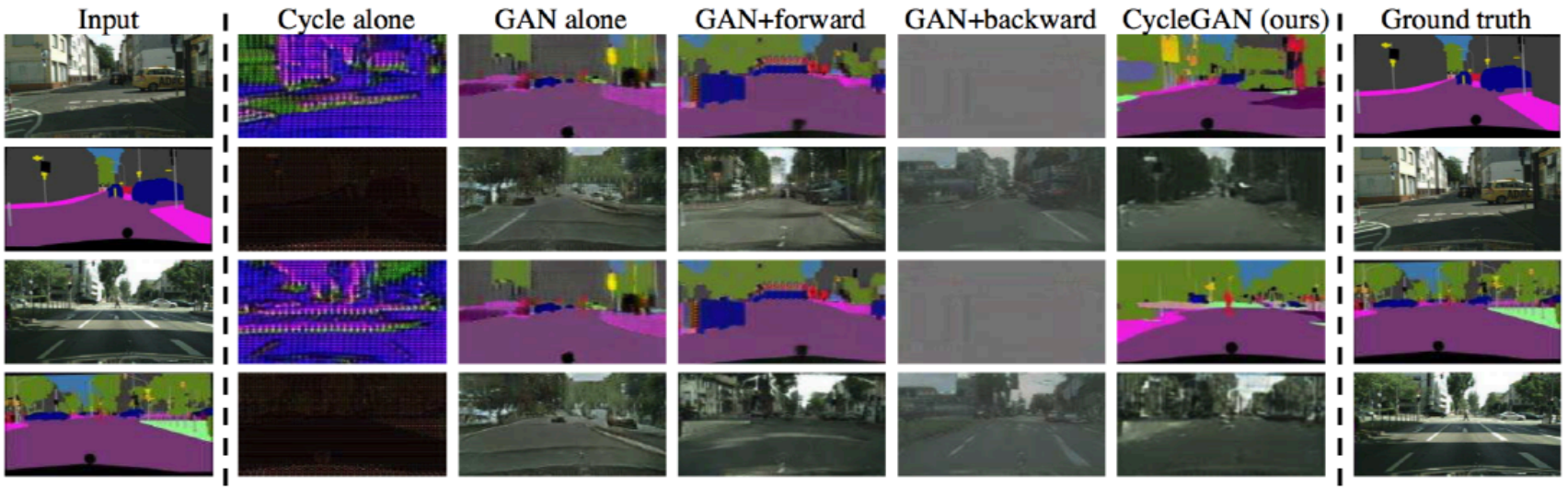
CycleGANs permit style transfer without matched training data

Style transfer between aerial photos and maps:



CycleGANs permit style transfer without matched training data

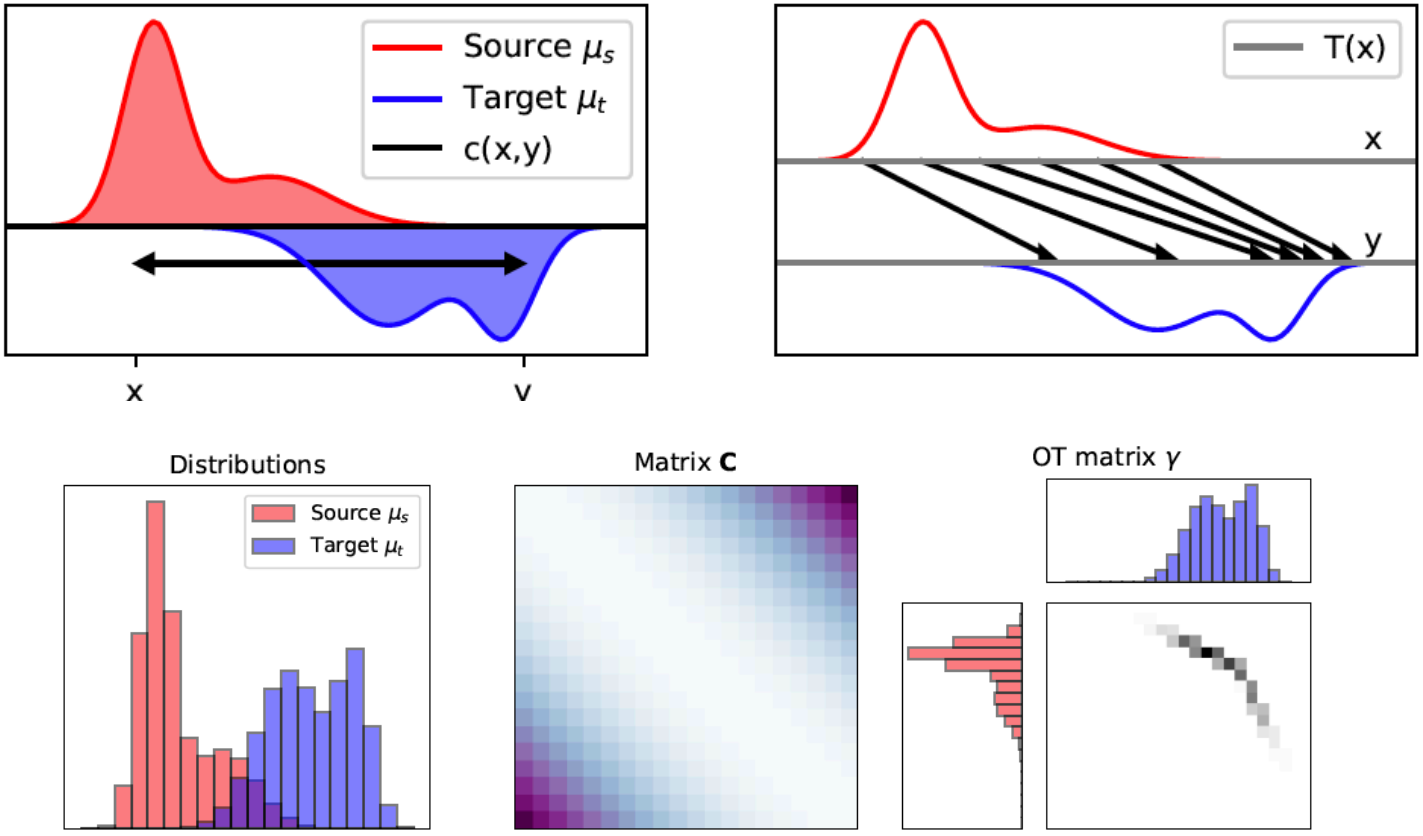
Style transfer between road scenes and semantic segmentations (labels of every pixel in an image by object category):



GANS with a Wasserstein loss (WGANs)

Wasserstein metric
Optimal Transport

Wasserstein metric = min Σ how much x how far



We can approximate the Wasserstein metric

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

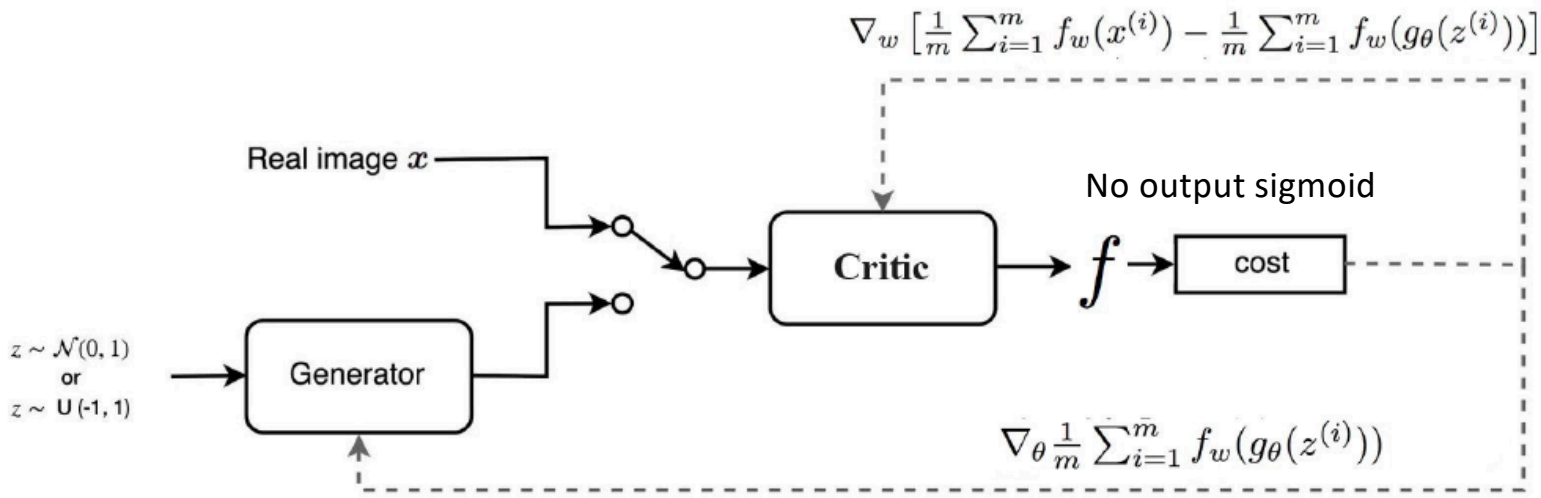
$$W(\mathbb{P}_r, \mathbb{P}_\theta) = \sup_{\|f\|_L \leq 1} \mathbb{E}_{x\sim\mathbb{P}_r}[f(x)] - \mathbb{E}_{x\sim\mathbb{P}_\theta}[f(x)]$$

$$|f(x_1) - f(x_2)| \leq |x_1 - x_2|.$$

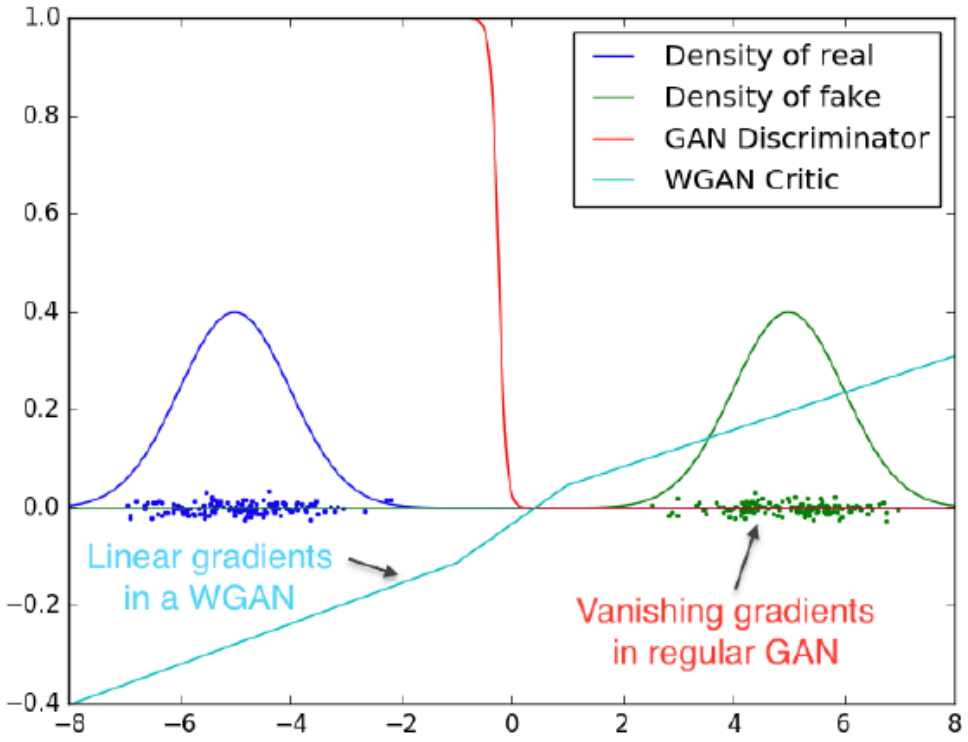
Metric can be approximated by a neural network directly; continuity constraint is maintained during training

$$w \leftarrow w + \alpha \cdot \mathrm{RMSProp}(w, g_w)$$
$$w \leftarrow \mathrm{clip}(w, -c, c)$$

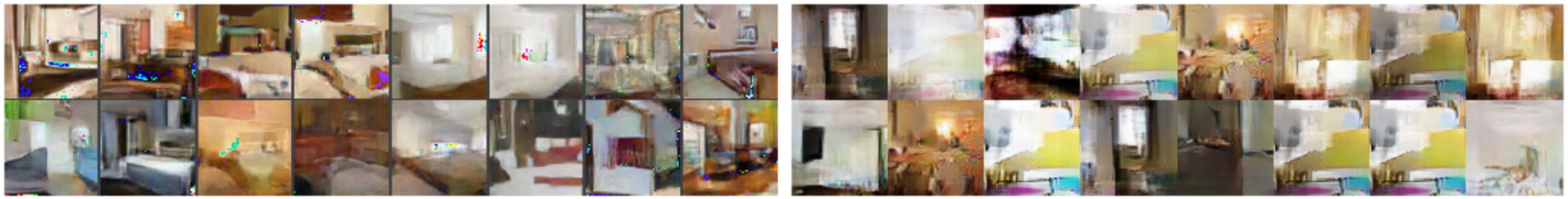
Wasserstein GAN (WGAN)



Wasserstein GAN (WGAN) gradients are better behaved



WGAN (left) GAN (right) samples from a MLP



https://arxiv.org/abs/1701.07875

FIN - Thank You