Generative Adversarial Networks

Mostly adapted from Goodfellow’s 2016 NIPS tutorial:

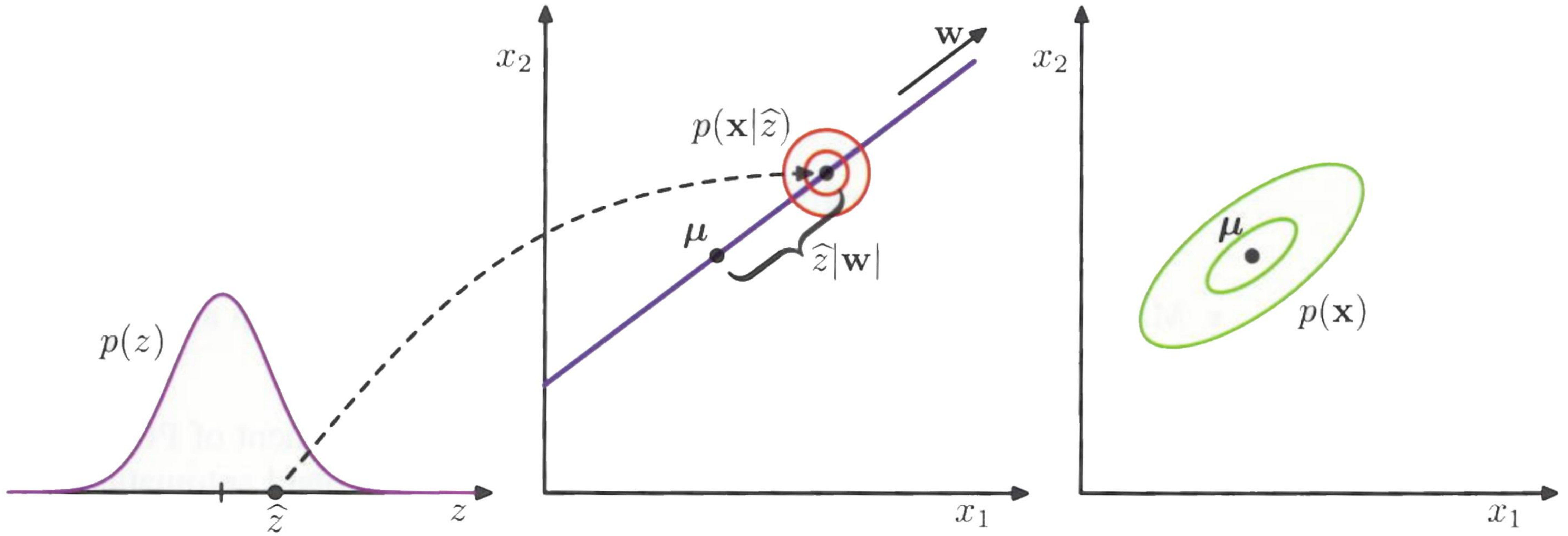
https://arxiv.org/pdf/1701.00160.pdf

Story so far: Why generative models?

- Unsupervised learning means we have more training data
- Some problems have many right answers, and diversity is desirable
  - Caption generation, image to image, super-resolution
- Some tasks intrinsically require generation
  - Machine translation
- Some generative models allow us to investigate a lower dimensional manifold of high dimensional data. This manifold can provide insight into high dimensional observations
  - Brain activity, gene expression

Recap: Factor Analysis

- Generative model: Assumes that data are generated from real valued latent variables



Bishop – Pattern Recognition and Machine Learning

Recap: Factor Analysis

- We can see from the marginal distribution:

$$p(\boldsymbol{x_i}|\boldsymbol{W},\boldsymbol{\mu},\boldsymbol{\Psi}) = \mathcal{N}(\boldsymbol{x_i}|\boldsymbol{\mu},\boldsymbol{\Psi}+\boldsymbol{W}\boldsymbol{W}^T)$$

that the covariance matrix of the data distribution is broken into 2 terms

- A diagonal part $\boldsymbol{\Psi}$: **variance not shared between variables**
- A low rank matrix $\boldsymbol{W}\boldsymbol{W}^T$: **shared variance due to latent factors**

Recap: Evidence Lower Bound (ELBO)

- From basic probability we have:

$$\mathrm{KL}\big(q(z) \,||\, p(z|x,\theta)\big) = \mathrm{KL}\big(q(z) \,||\, p(x,z\,|\theta)\big) + \log p(x|\theta)$$

- We can rearrange the terms to get the following decomposition:

$$\log p(x|\theta) = \mathrm{KL}\big(q(z) \,||\, p(z|x,\theta)\big) - \mathrm{KL}\big(q(z) \,||\, p(x,z\,|\theta)\big)$$

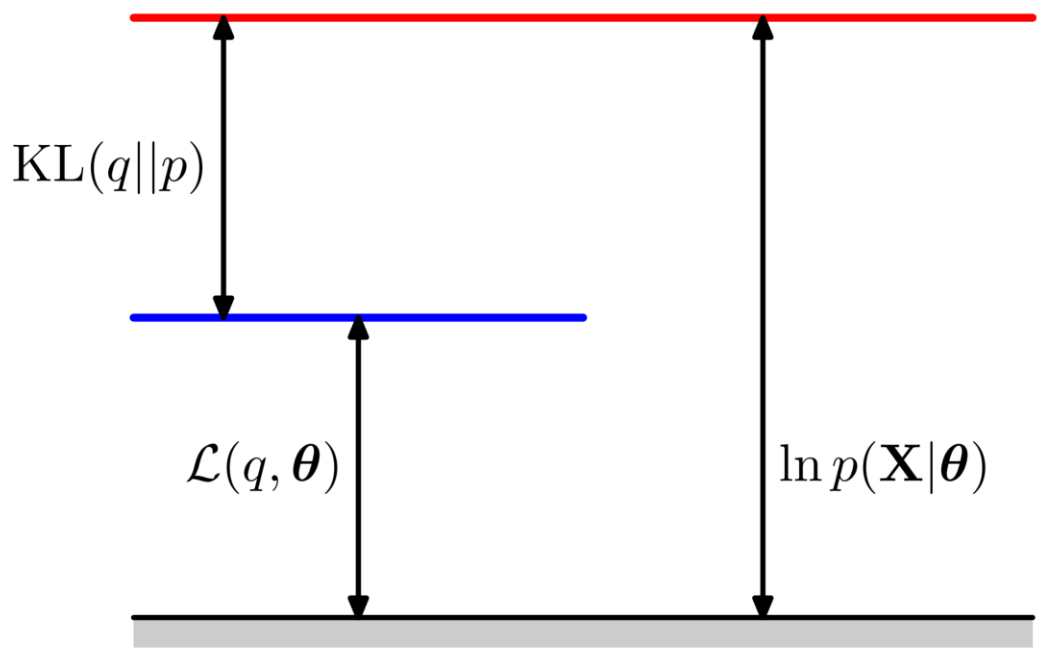
- We define the *evidence lower bound* (ELBO) as:

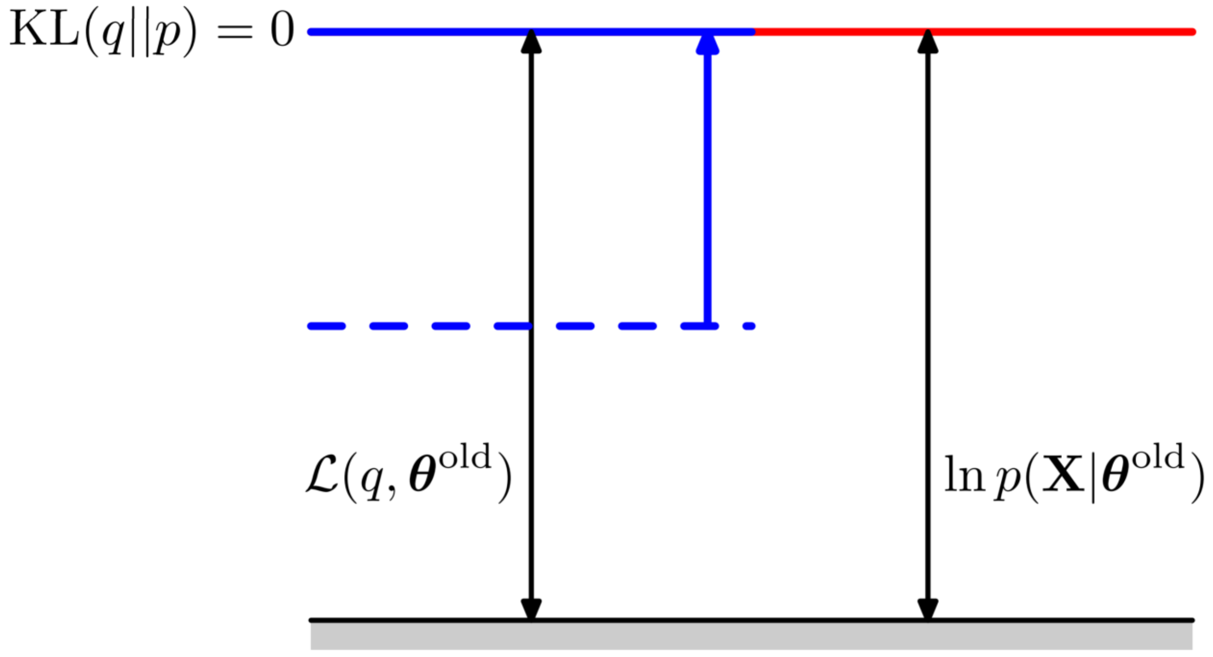
$$\mathcal{L}(q,\theta) \triangleq -\mathrm{KL}\big(q(z) \,||\, p(x,z\,|\theta)\big)$$

Then:

$$\log p(x|\theta) = \mathrm{KL}\big(q(z)||p(z|x,\theta)\big) + \mathcal{L}(q,\theta)$$

Recap: The EM algorithm E step

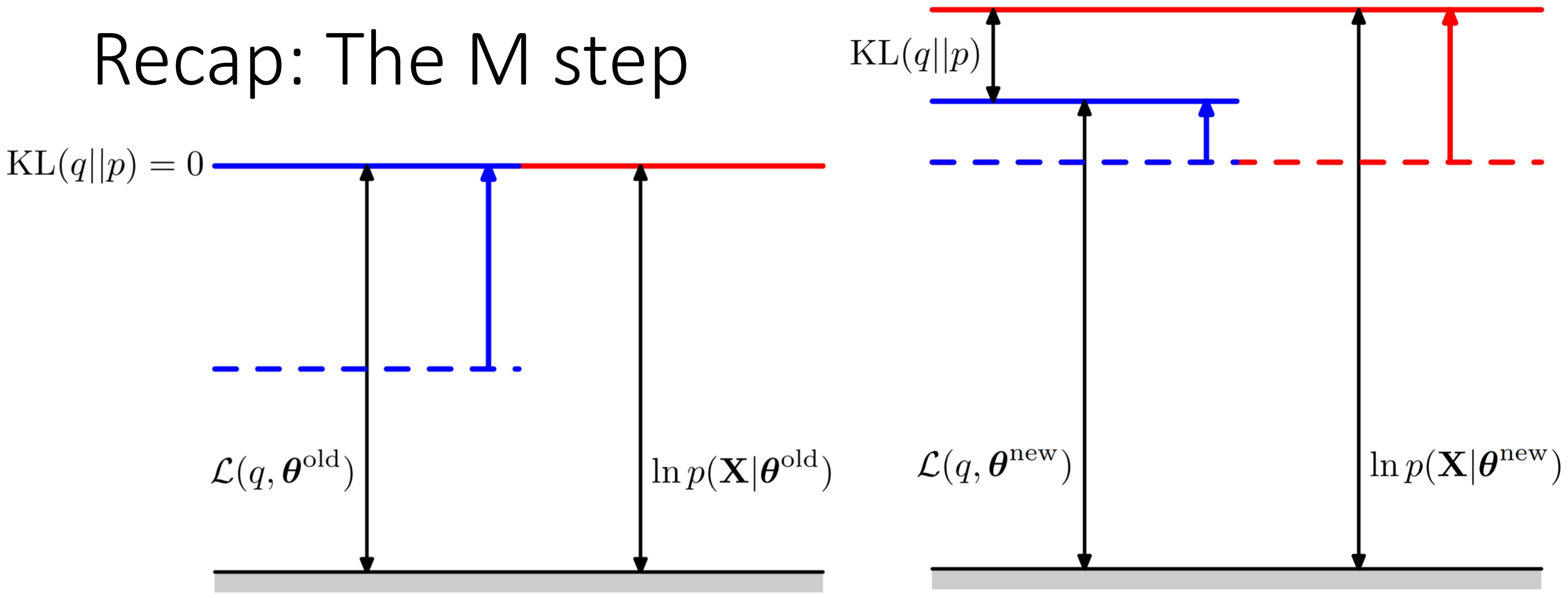

$\mathrm{KL}(q||p)$
$\mathcal{L}(q,\boldsymbol{\theta})$
$\ln p(\mathbf{X}|\boldsymbol{\theta})$


$\mathrm{KL}(q||p)=0$
$\mathcal{L}(q,\boldsymbol{\theta}^{\mathrm{old}})$
$\ln p(\mathbf{X}|\boldsymbol{\theta}^{\mathrm{old}})$

Bishop – Pattern Recognition and Machine Learning

- Maximize $\mathcal{L}\left(q,\theta^{(t-1)}\right)$ with respect to q by setting $\boldsymbol{q}^{(t)}(\boldsymbol{z}) \leftarrow \boldsymbol{p}\left(\boldsymbol{z}\middle|\boldsymbol{x},\boldsymbol{\theta}^{(t-1)}\right)$

Recap: The M step



$\mathrm{KL}(q||p) = 0$
$\mathcal{L}(q, \boldsymbol{\theta}^{\mathrm{old}})$
$\ln p(\mathbf{X}|\boldsymbol{\theta}^{\mathrm{old}})$
$\mathrm{KL}(q||p)$
$\mathcal{L}(q, \boldsymbol{\theta}^{\mathrm{new}})$
$\ln p(\mathbf{X}|\boldsymbol{\theta}^{\mathrm{new}})$

Bishop – Pattern Recognition and Machine Learning

- After applying the E step, we increase the likelihood of the data by finding better parameters according to: $\theta^{(t)} \leftarrow \mathbf{argmax}_{\boldsymbol{\theta}}\, \mathbb{E}_{\boldsymbol{q}^{(t)}(\boldsymbol{z})}[\mathbf{log}\, \boldsymbol{p}(\boldsymbol{x}, \boldsymbol{z}\, |\boldsymbol{\theta})]$

Recap: EM in practice

$$\operatorname{argmax}_{\boldsymbol{W},\boldsymbol{\Psi}} \mathbb{E}_{q^{(t)}(\boldsymbol{z})}[\log p(\boldsymbol{X},\boldsymbol{Z}\,|\boldsymbol{W},\boldsymbol{\Psi})] =$$

$$= \operatorname{argmax}_{\boldsymbol{W},\boldsymbol{\Psi}} -\frac{N}{2}\log\det(\boldsymbol{\Psi}) - \sum_{i=1}^{N}\left(\frac{1}{2}\boldsymbol{x}_i^T\boldsymbol{\Psi}^{-1}\boldsymbol{x}_i - \boldsymbol{x}_i^T\boldsymbol{\Psi}^{-1}\boldsymbol{W}\mathbb{E}_{q^{(t)}(\boldsymbol{z}_i)}[\boldsymbol{z}_i]\right.$$

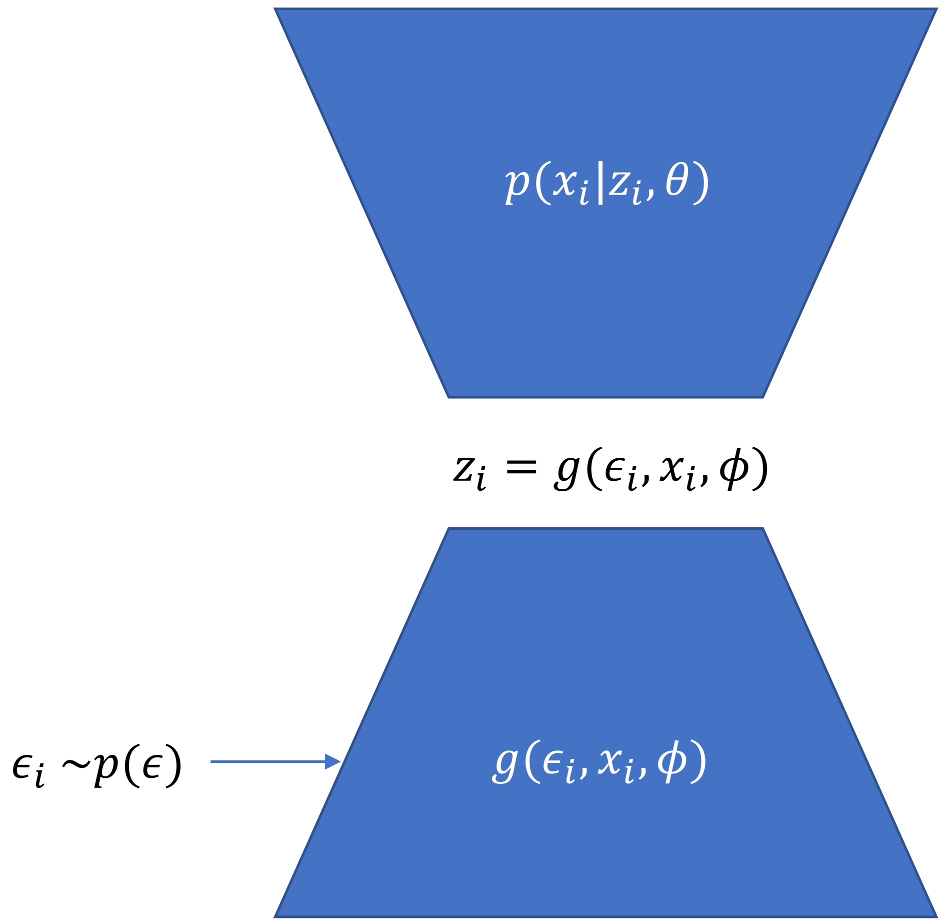
$$\left.+\frac{1}{2}\operatorname{tr}\left(\boldsymbol{W}^T\boldsymbol{\Psi}^{-1}\boldsymbol{W}\mathbb{E}_{q^{(t)}(\boldsymbol{z}_i)}\left[\boldsymbol{z}_i\boldsymbol{z}_i^T\right]\right)\right)$$

- **By looking at what expectations the M step requires, we find out what we need to compute in the E step.**
- **For FA, we only need these 2 sufficient statistics to enable the M step.**
- **In practice, sufficient statistics are often what we compute in the E step**

Recap: From EM to Variational Inference

- In EM we alternately maximize the ELBO with respect to θ and probability distribution (functional) q
- In variational inference, we **drop the distinction between hidden variables and parameters** of a distribution
- I.e. we replace $p(x, z|\theta)$ with $p(x, z)$. Effectively this puts a **probability distribution on the parameters $\boldsymbol{\theta}$**, then absorbs them into z
- Fully Bayesian treatment instead of a point estimate for the parameters

Recap: Variational Autoencoder


$p(x_i|z_i,\theta)$
$z_i = g(\epsilon_i, x_i, \phi)$
$\epsilon_i \sim p(\epsilon)$
$g(\epsilon_i, x_i, \phi)$

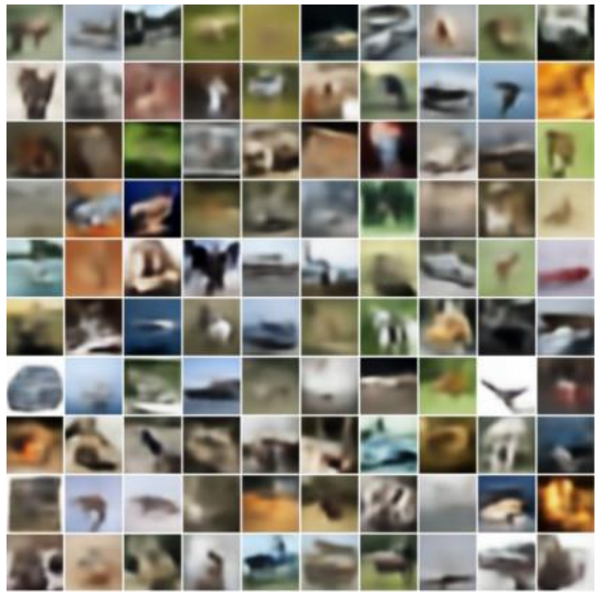
- For $t = 1:b:T$
    - Estimate $\frac{\partial \mathcal{L}}{\partial \phi}, \frac{\partial \mathcal{L}}{\partial \theta}$ with either $-\tilde{\mathcal{L}}^A$ or $-\tilde{\mathcal{L}}^B$ as the loss
    - Update ϕ, θ

- Training procedure uses standard back propagation with an MC procedure to approximately run EM on the ELBO
- The reparameterization trick enables the gradient to flow through the network

Recap: Requirements of the VAE

- Note that the VAE requires 2 tractable distributions to be used:
  - The prior distribution $p(z)$ must be easy to sample from
  - The conditional likelihood $p(x|z, \theta)$ must be computable
- In practice this means that the 2 distributions of interest are often simple, for example uniform, Gaussian, or even isotropic Gaussian

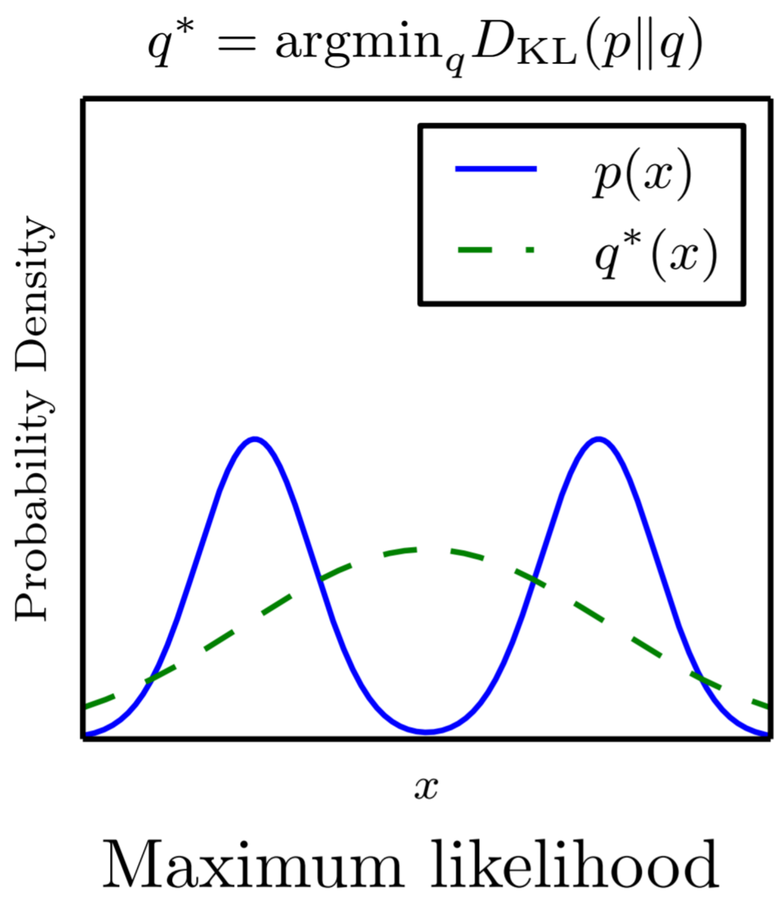
Recap: The VAE blurry image problem

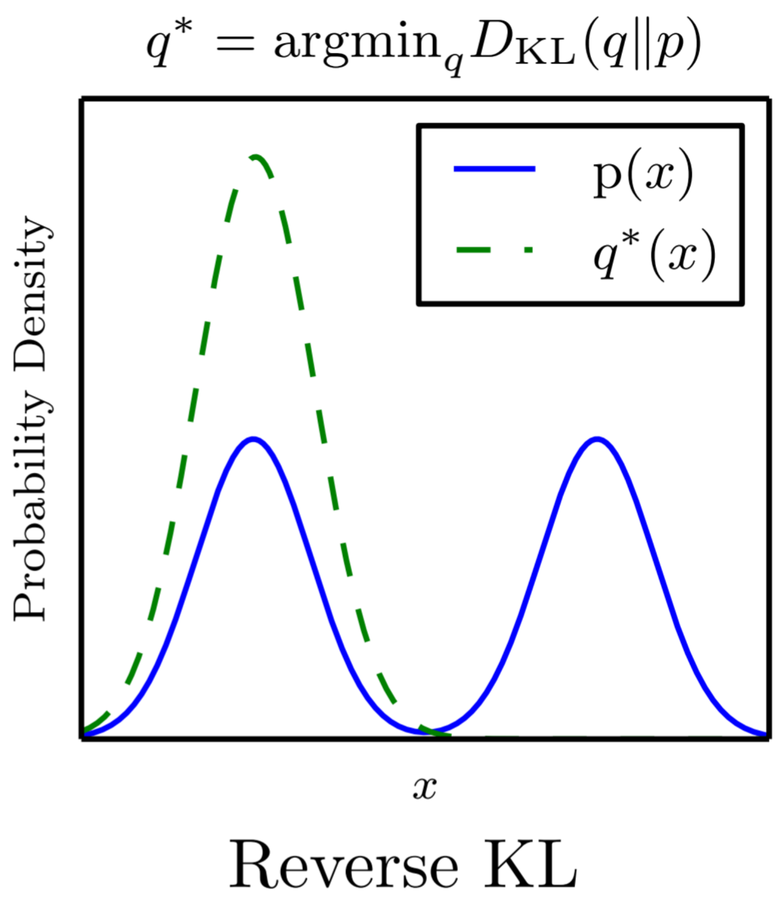


https://blog.openai.com/generative-models/

- The samples from the VAE look blurry
- Three plausible explanations for this
  - Maximizing the likelihood
  - **Restrictions on the family of distributions**
  - **The lower bound approximation**

Recap: The maximum likelihood explanation

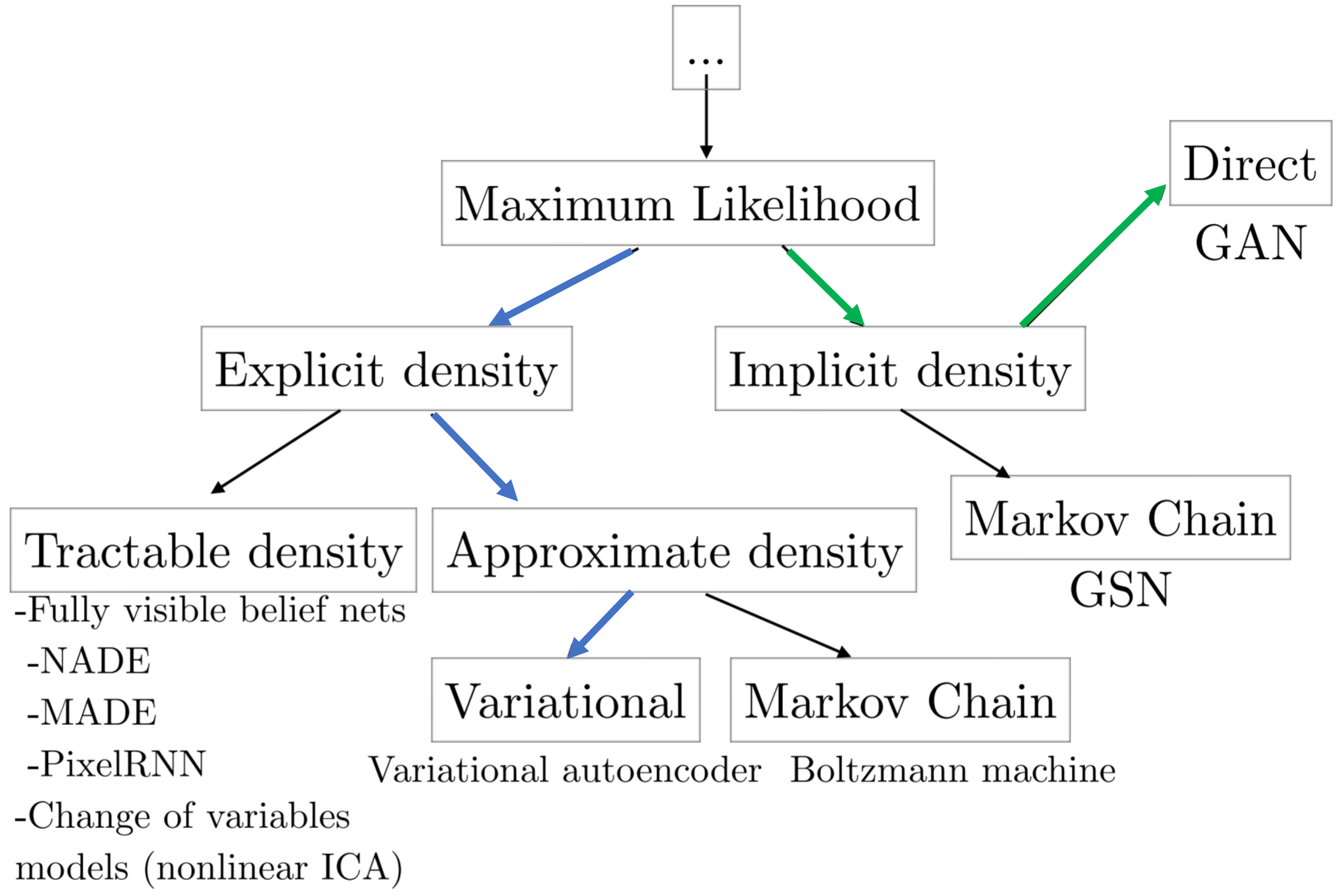




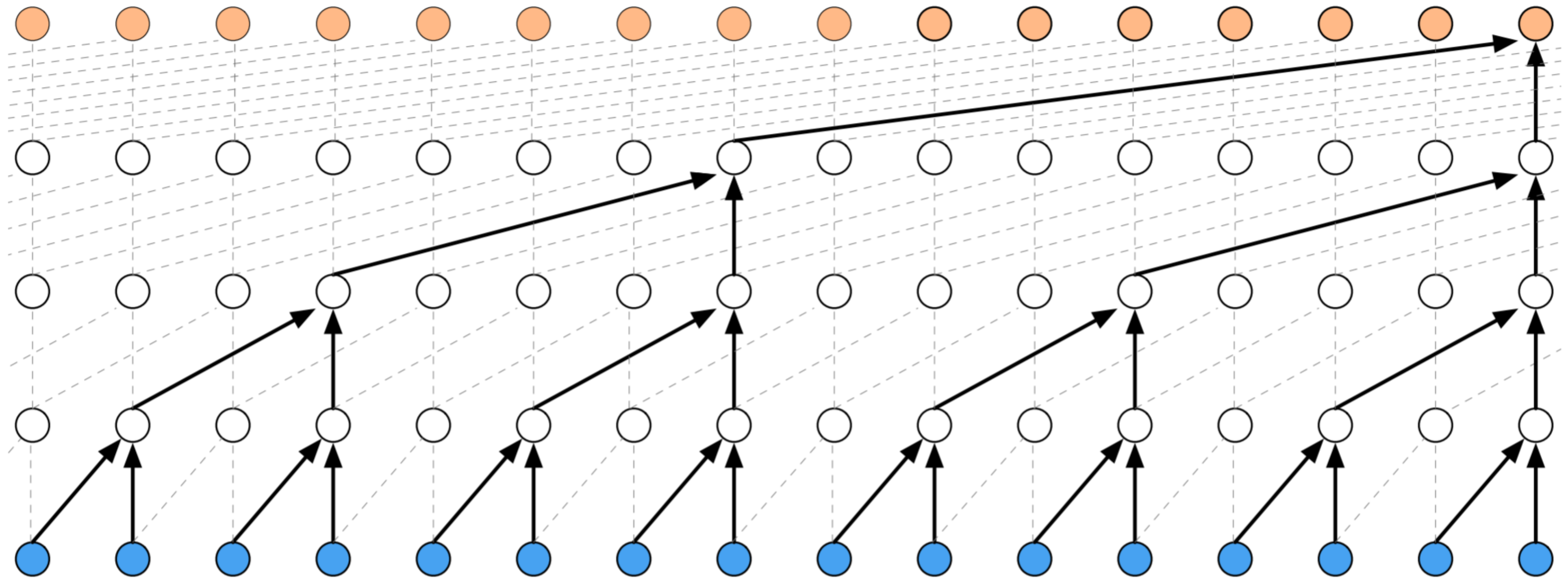
- Recent evidence suggests that this is not actually the problem
- **GANs can be trained with maximum likelihood and still generate sharp examples**

https://arxiv.org/pdf/1701.00160.pdf

A taxonomy of generative models



Fully Visible Belief Net (FVBN), e.g. Wavenet



$$p(\boldsymbol{x}) = \prod_{t=1}^{T} p(x_t | x_1, \dots, x_{t-1})$$

- No latent variable (hence fully visible)
- Tractable log-likelihood
- Train with auto-regressive target
- Easier to optimize well
- Slower to run

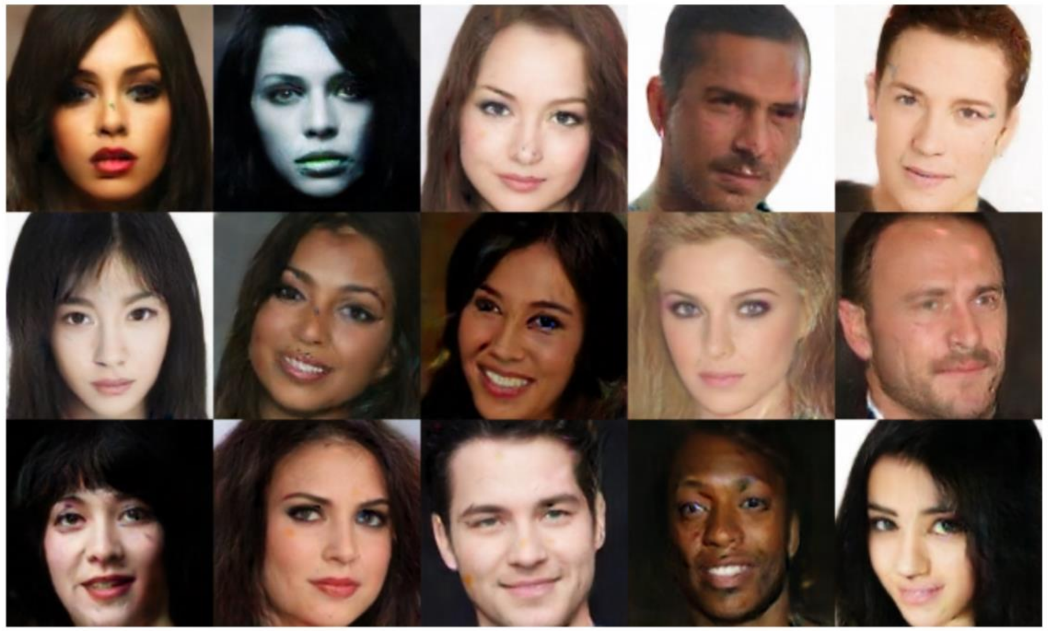
GAN Advantages

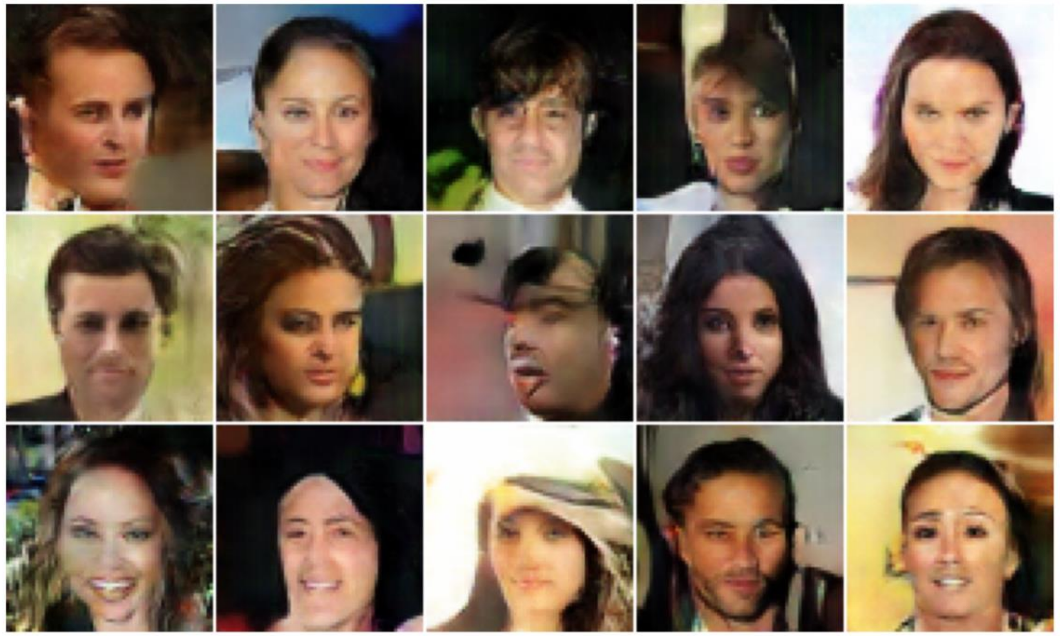
- Sample in parallel (vs FVBN)
- Few restrictions on generator function
- No Markov Chain
- No variational bound
- Subjectively better samples

GAN Disadvantages

- Very difficult to train properly
- Difficult to evaluate
- Likelihood cannot be computed
- No encoder (in vanilla GAN)

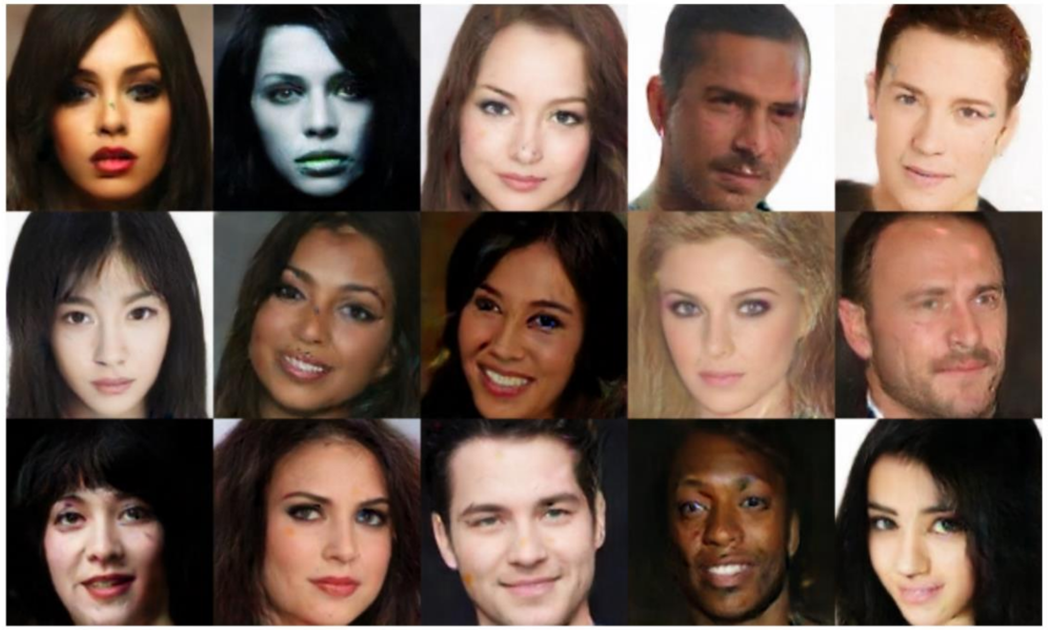
GAN samples look sharp

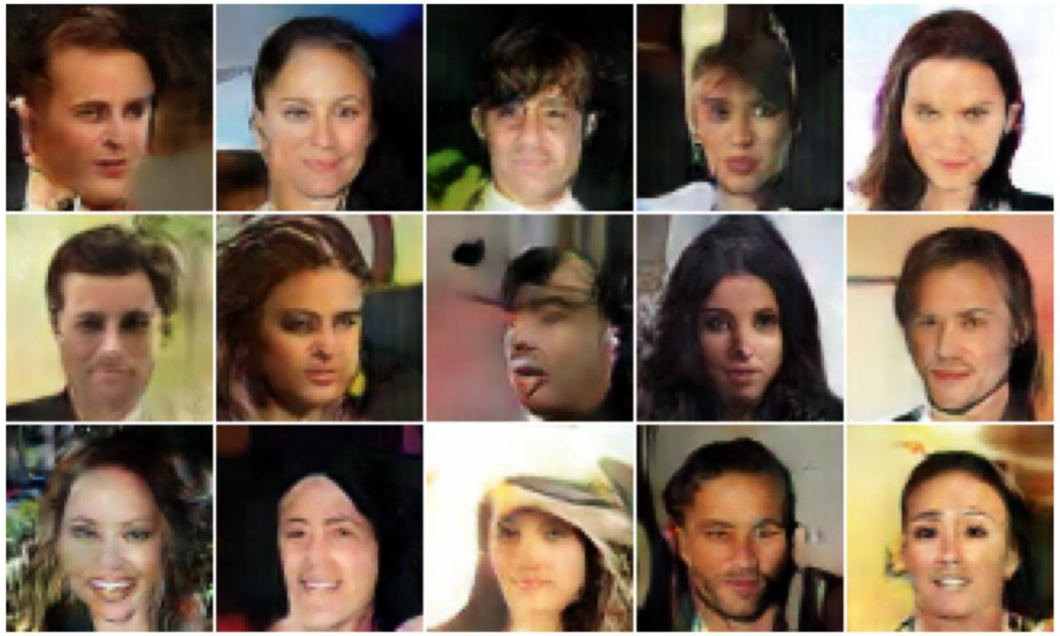

Real Samples


Generated Samples

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

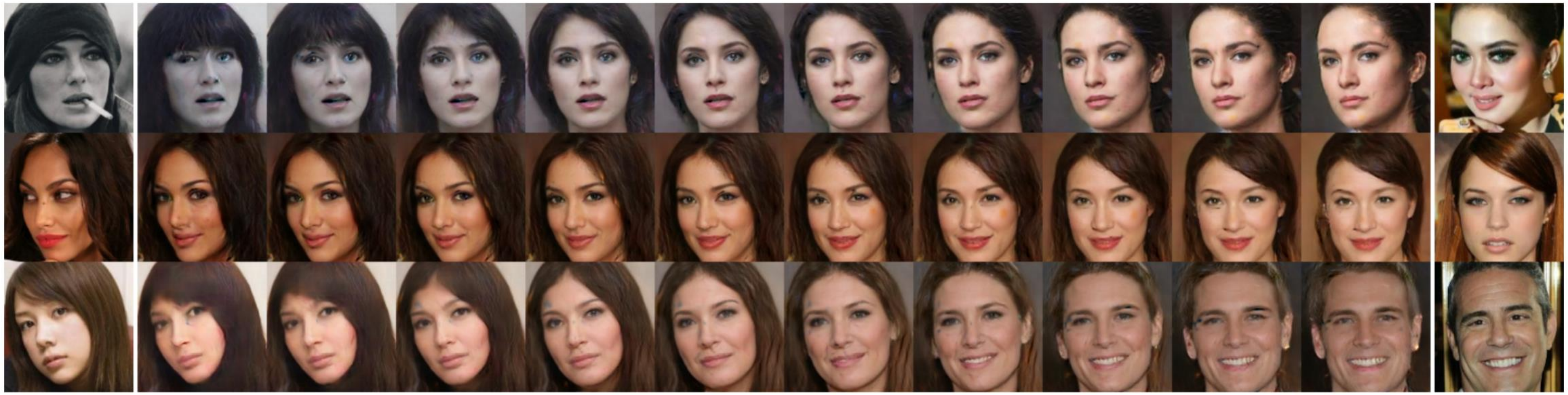
GAN samples look sharp

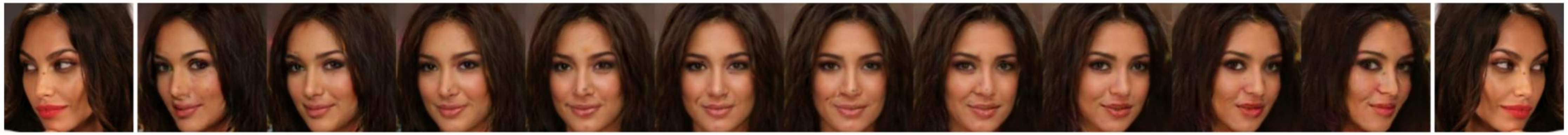

~~Real Samples~~
Boundary Equilibrium GAN


~~Generated Samples~~
Energy Based GAN

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

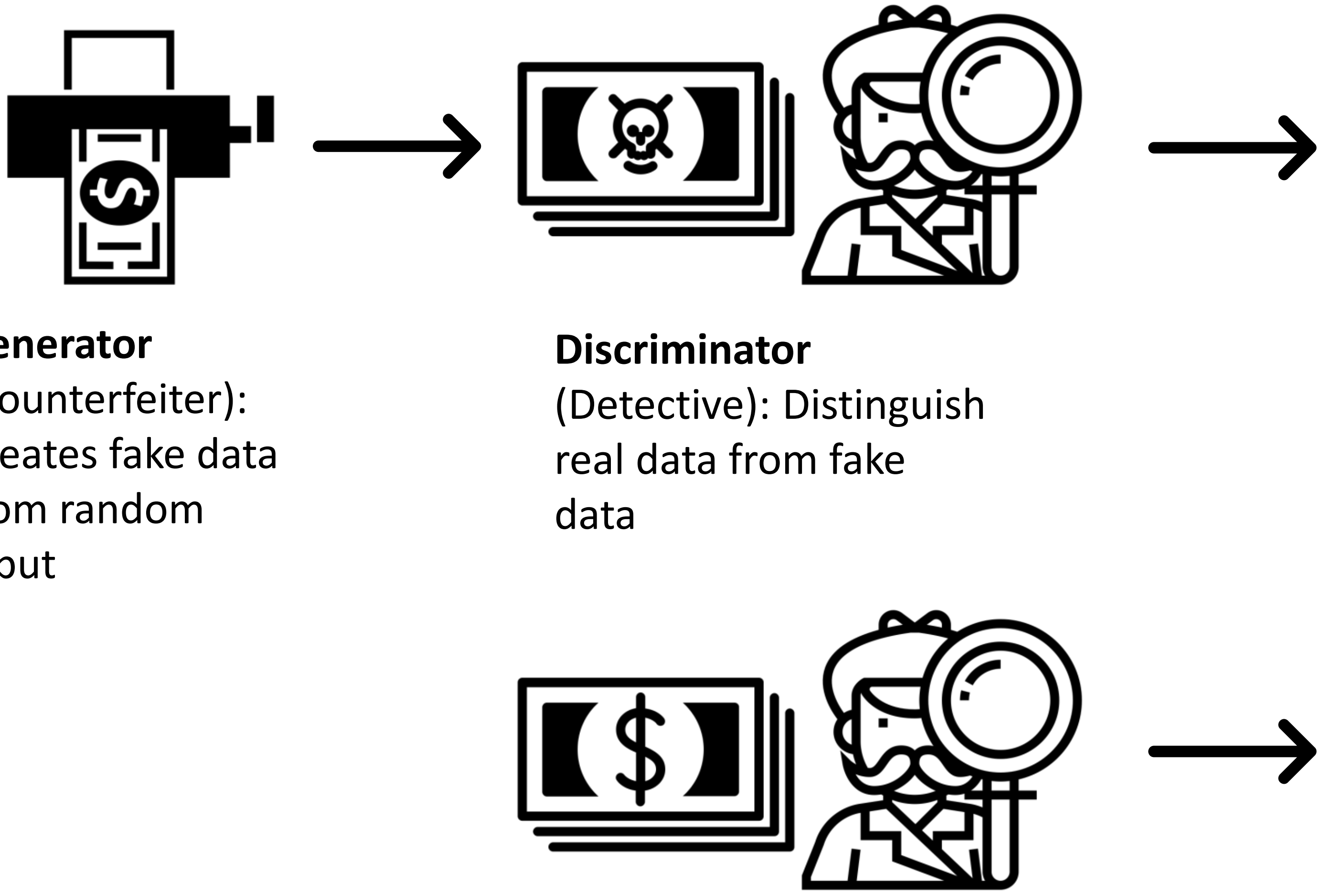
Interpolation is impressive


(c) Our results (128x128 with 128 filters)


(d) Mirror interpolations (our results 128x128 with 128 filters)

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

Generative Adversarial Networks: Basic idea



Looks Fake!

Generator (Counterfeiter): Creates fake data from random input

Discriminator (Detective): Distinguish real data from fake data

Looks Real!

The Generator

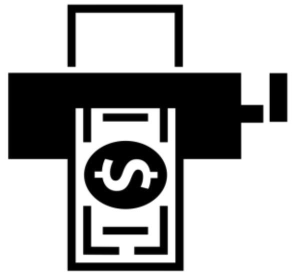
- Faking Data
  - To create good fake data, the generator must understand what real data looks like
  - **Attempts to generate samples that are likely under the true data distribution**
  - **Implicitly** learns to **model the true distribution**
- Latent Code
  - Since the sample is determined by the random noise input, the probability distribution is conditioned on this input
  - The random noise is **interpreted by the model as a latent code**, i.e. a point on the manifold

Problem setup

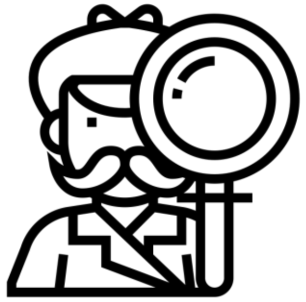
Generator Trained to get better and better at fooling the discriminator (making fake data look real)

Discriminator Trained to get better and better at distinguishing real data from fake data

Formalizing the generator/discriminator

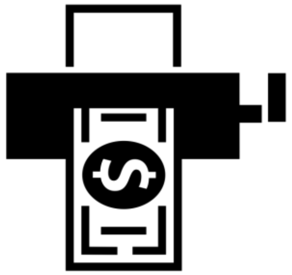


Generator: $G\left(z, \theta^{(G)}\right)$
A differentiable function, G (here having parameters $\theta^{(G)}$), mapping from the latent space, $\mathbb{R}^L$, to the data space, $\mathbb{R}^M$



Discriminator: $D\left(x, \theta^{(D)}\right)$
A differentiable function, D (here having parameters $\theta^{(D)}$), mapping from the data space, $\mathbb{R}^M$, to a scalar between 0 and 1 representing the probability that the data is real

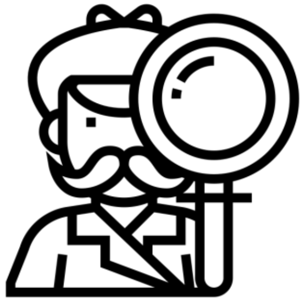
Simplifying notation



Generator: $G(z)$
For simplicity of notation, we write $G(z)$ without $\theta^{(G)}$

Typically G is a neural network, but it doesn't have to be

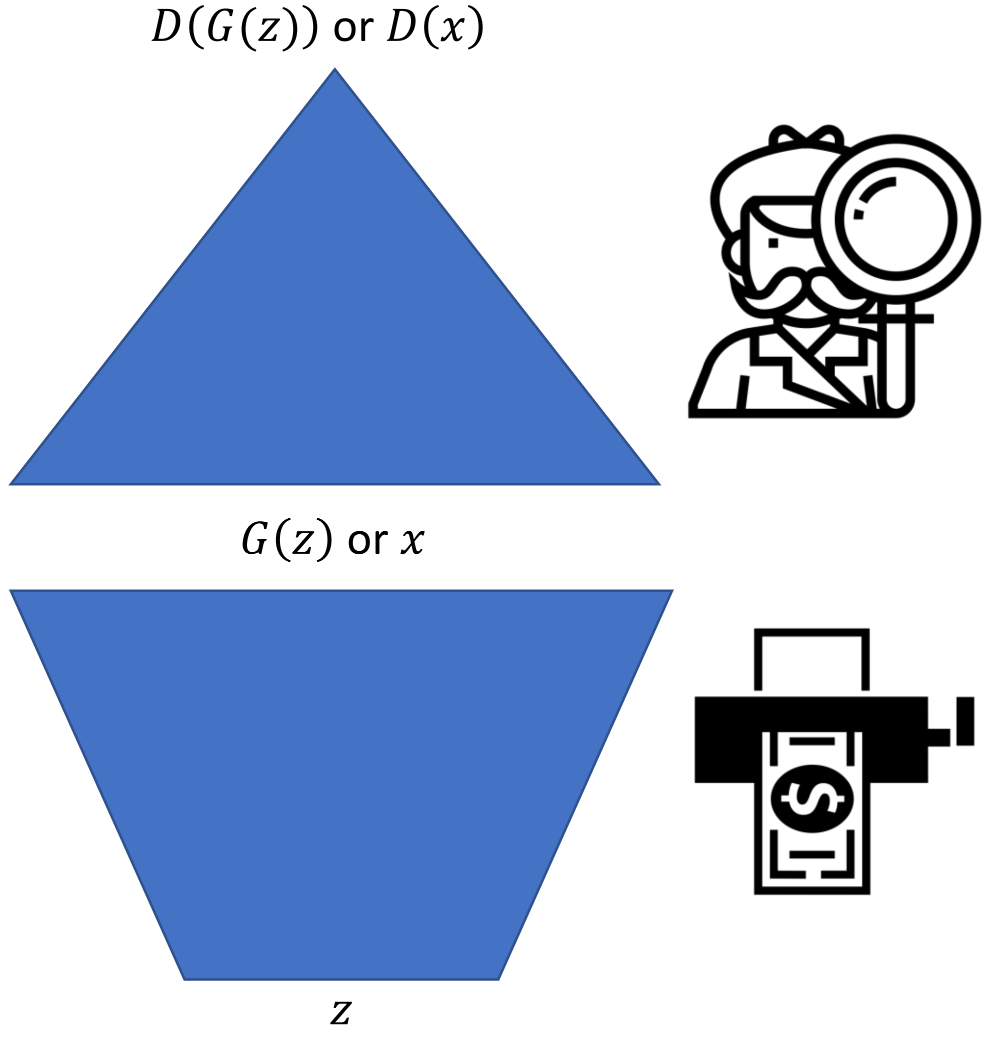
Note z can go into any layer of the network, not just the first



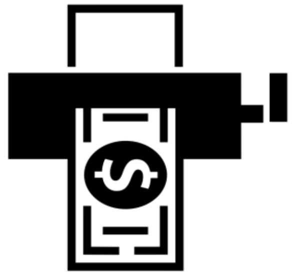
Discriminator: $D(x), D(G(z))$
Note that the discriminator can also take the output of the generator as input.

Typically D is a neural network, but it doesn't have to be

An artist's rendition



The game (theory)



- The generator and discriminator are adversaries in a game
- The generator controls only its parameters
- The discriminator controls only its parameters
- Each seeks to maximize its own success and minimize the success of the other: related to **minimax** theory

Nash equilibrium

- In game theory, a local optimum in this system is called a Nash equilibrium:
- Generator loss, $J^{(G)}$, is at a local minimum with respect to $\theta^{(G)}$
- Discriminator loss, $J^{(D)}$, is at a local minimum with respect to $\theta^{(D)}$

Basic training procedure

```
• Initialize $\theta^{(G)}, \theta^{(D)}$
• For $t = 1{:}\, b{:}\, T$
    • Initialize $\Delta\theta^{(D)} = 0$
    • For $i = t{:}\, t + b - 1$
        • Sample $z_i \sim p(z_i)$
        • Compute $D(G(z_i)), D(x_i)$
        • $\Delta\theta_i^{(D)} \leftarrow$ Compute gradient of **Discriminator loss**, $J^{(D)}(\theta^{(G)}, \theta^{(D)})$
        • $\Delta\theta^{(D)} \leftarrow \Delta\theta^{(D)} + \Delta\theta_i^{(D)}$
    • Update $\theta^{(D)}$
    • Initialize $\Delta\theta^{(G)} = 0$
    • For $j = t{:}\, t + b - 1$
        • Sample $z_j \sim p(z_j)$
        • Compute $D\left(G(z_j)\right), D(x_j)$
        • $\Delta\theta_j^{(G)} \leftarrow$ Compute gradient of **Generator loss,** $J^{(G)}(\theta^{(G)}, \theta^{(D)})$
        • $\Delta\theta^{(G)} \leftarrow \Delta\theta^{(G)} + \Delta\theta_j^{(G)}$
    • Update $\theta^{(G)}$
```

Can also run k minibatches of the discriminator update before updating the generator, but Goodfellow finds $k = 1$ tends to work best

Basic training procedure

```
• Initialize $\theta^{(G)}, \theta^{(D)}$
• For $t = 1: b: T$
    • Initialize $\Delta\theta^{(D)} = 0$
    • For $i = t: t + b - 1$
        • Sample $z_i \sim p(z_i)$
        • Compute $D(G(z_i)), D(x_i)$
        • $\Delta\theta_i^{(D)} \leftarrow$ Compute gradient of **Discriminator loss**, $J^{(D)}(\theta^{(G)}, \theta^{(D)})$
        • $\Delta\theta^{(D)} \leftarrow \Delta\theta^{(D)} + \Delta\theta_i^{(D)}$
    • Update $\theta^{(D)}$
    • Initialize $\Delta\theta^{(G)} = 0$
    • For $j = t: t + b - 1$
        • Sample $z_j \sim p(z_j)$
        • Compute $D\left(G(z_j)\right), D(x_j)$
        • $\Delta\theta_j^{(G)} \leftarrow$ Compute gradient of **Generator loss,** $J^{(G)}(\theta^{(G)}, \theta^{(D)})$
        • $\Delta\theta^{(G)} \leftarrow \Delta\theta^{(G)} + \Delta\theta_j^{(G)}$
    • Update $\theta^{(G)}$
```

Notice: the only explicit probability distribution we have is the random noise distribution, the prior

The loss causes the data distribution to be learned implicitly

Simplified training procedure

```
• Initialize $\theta^{(G)}, \theta^{(D)}$
• For $t = 1: b: T$
    • Initialize $\Delta\theta^{(G)} = \Delta\theta^{(D)} = 0$
    • For $i = t: t + b - 1$
        • Sample $z_i \sim p(z_i)$
        • Compute $D\big(G(z_i)\big), D(x_i)$
        • $\Delta\theta_i^{(D)} \leftarrow$ Compute $\partial_{\theta^{(D)}} J^{(D)}\big(\theta^{(G)}, \theta^{(D)}\big)$
        • $\Delta\theta_j^{(G)} \leftarrow$ Compute $\partial_{\theta^{(G)}} J^{(G)}\big(\theta^{(G)}, \theta^{(D)}\big)$
        • $\Delta\theta^{(D)} \leftarrow \Delta\theta^{(D)} + \Delta\theta_i^{(D)}$
        • $\Delta\theta^{(G)} \leftarrow \Delta\theta^{(G)} + \Delta\theta_j^{(G)}$
    • Update $\theta^{(D)}, \theta^{(G)}$
```

Update the discriminator and generator from the same pair of mini-batches

Discriminator (D)'s loss function

$$J^{(D)}\left(\theta^{(D)}, \theta^{(G)}\right)$$
$$= -\frac{1}{2}\mathbb{E}_{x\sim p_{data}}[\log D(x)] - \frac{1}{2}\mathbb{E}_{z\sim p_z}\left[\log\left(1 - D\left(G(z)\right)\right)\right]$$

- **Binary cross-entropy** (almost)
- The first term is for real data (positive classification)
- The second term is for fake data (negative classification)
- Differs from cross-entropy only in what we take the expectation over
- **Supervised** loss on data with **no labels**

Generator (G)'s loss function

- Take the negative of the discriminator's loss:

$$J^{(G)}\left(\theta^{(D)},\theta^{(G)}\right) = -J^{(D)}\left(\theta^{(D)},\theta^{(G)}\right)$$

- With this loss, we have a **value function** describing a **zero-sum game:**

$$\min_{\boldsymbol{G}} \max_{\boldsymbol{D}} -J^{(D)}\left(\theta^{(D)},\theta^{(G)}\right)$$

- Attractive to analyze with game theory
- There is a problem with this loss for gradient descent (we'll come back to this)

Rewriting $J^{(D)}$

$$J^{(D)}(\theta^{(D)}, \theta^{(G)}) = -\frac{1}{2}\mathbb{E}_{x\sim p_{data}} \log D(x) - \frac{1}{2}\mathbb{E}_z \log\left(1 - D\left(G(z)\right)\right)$$

$$= -\frac{1}{2}\left[\int_x p_{data}(x) \log D(x) dx + \int_z p_z(z) \log\left(1 - D\left(G(z)\right)\right) dz\right]$$

$$= -\frac{1}{2}\left[\int_x p_{data}(x) \log D(x) + p_G(x) \log(1 - D(x)) dx\right]$$

Optimal discriminator

$$J^{(D)}\left(\theta^{(D)}, \theta^{(G)}\right) = -\frac{1}{2}\left[\int_x p_{data}(x) \log D(x) + p_G(x) \log\left(1 - D(x)\right) dx\right]$$

Take the functional derivative w.r.t. $D(x)$ and set to 0, analogous to:

$$\frac{\partial}{\partial y}\left(p_{data}(x) \log y + p_G(x) \log(1 - y)\right) = 0$$

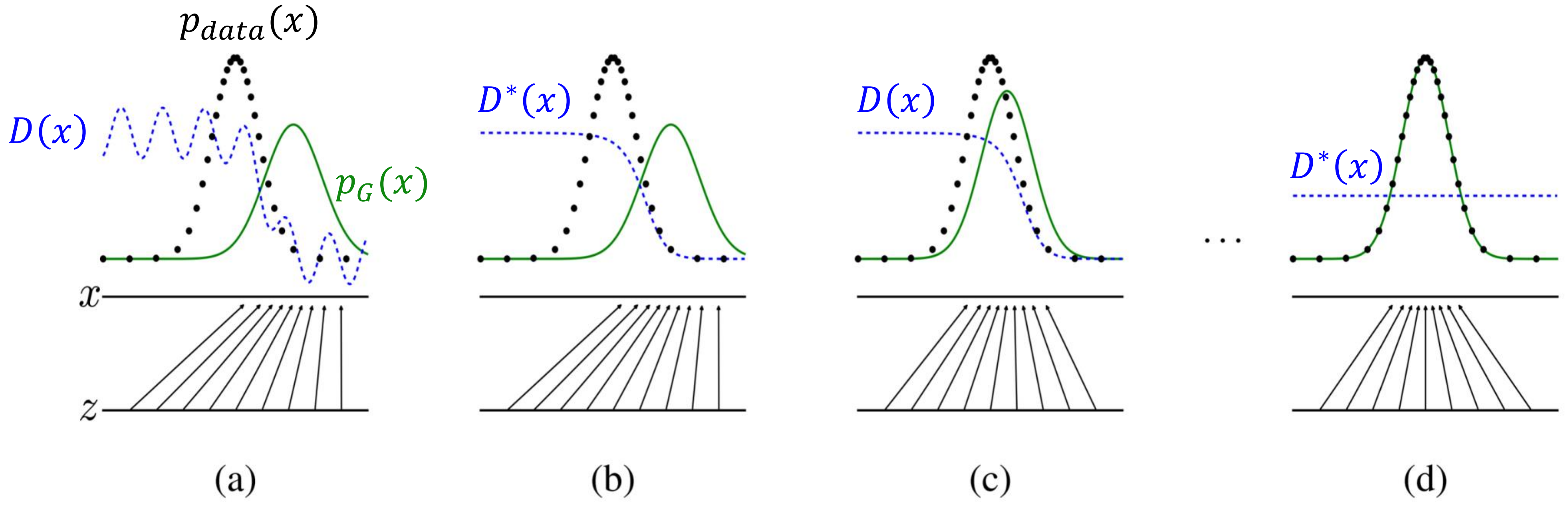
$$\frac{p_{data}(x)}{y} - \frac{p_G(x)}{1 - y} = 0$$

$$y = \frac{p_{data}(x)}{p_{data}(x) + p_G(x)} \rightarrow D^*(x) = \frac{p_{data}(x)}{p_{data}(x) + p_G(x)}$$

- We are assuming that $p_{data}(x)$, $p_G(x)$ are non-zero everywhere

Optimal discriminator

- The best strategy for the discriminator is to learn the ratio of the probabilities of x under the data distribution and the generator distribution: $D^*(x) = \frac{p_{data}(x)}{p_{data}(x)+p_G(x)} = p(data|x)$

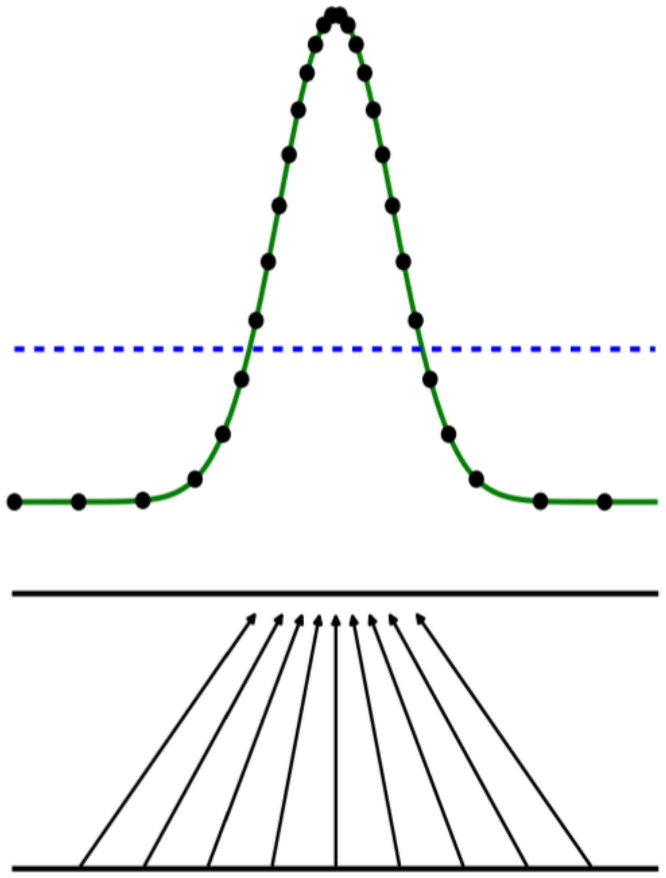


Discriminator intuition

$$J^{(D)}\left(\theta^{(D)}, \theta^{(G)}\right) = -\frac{1}{2}\mathbb{E}_{x\sim p_{data}} \log D(x) - \frac{1}{2}\mathbb{E}_z \log\left(1 - D\left(G(z)\right)\right)$$

- With this loss, **the discriminator approximates the ratio of $\frac{p_{data}(x)}{p_G(x)}$ via supervised learning**

Optimal generator



- With a few more steps, we can show that the global optimum for this game is achieved if and only if $p_G(x) = p_{data}(x)$
- We are, in theory, minimizing the Jensen-Shannon divergence between the generator distribution and the true data distribution!

Getting to the optimum

- For models **that have enough capacity**, if we use $J^{(G)} = -J^{(D)}$, and if D is **set to its global optimum given G** at every iteration and G **improves the criterion** at every iteration, then alternating optimization will get us to the global optimum
- In practice:
  - D, G may not have enough capacity
  - We do not get to find the global optimum for D at each iteration
- Theory tells us we want the discriminator to always be strong (in practice, there may be reasons to weaken it)

More gaps between theory and practice

- The theory assumes we can reach a global optimum
  - We have functions which are non-convex in the parameters we are optimizing: $J^{(D)}\left(\theta^{(D)}, \theta^{(G)}\right), J^{(G)}\left(\theta^{(D)}, \theta^{(G)}\right)$
- The theory assumes that $p_G(x), p_{data}(x)$ are non-zero everywhere. This may not hold – especially if we have data lying on a manifold. Even when it holds the ratio can be numerically unstable
- The theory assumes that the optimal discriminator is unique. In practice other discriminators can do nearly as good a job: i.e. the discriminator can overfit the data

Theory summary

- The theory gives us some insight into what GANs are doing
- Many of the assumptions in the theory do not hold
- We cannot get to the global optimum
- It can be difficult to even get to a local optimum
- Optimizing GANs is an active area of research (and the subject of much of today)

A problem with $J^{(G)} = -J^{(D)}$

- Setting $J^{(G)} = -J^{(D)}$, we have:

$$J^{(G)}\left(\theta^{(D)}, \theta^{(G)}\right) = \frac{1}{2}\mathbb{E}_{x \sim p_{data}} \log D(x) + \frac{1}{2}\mathbb{E}_{z} \log\left(1 - D\left(G(z)\right)\right)$$

- What happens to the second term when the discriminator is much better than the generator?

$$D\left(G(z)\right) \to 0$$

$$\frac{1}{2}\mathbb{E}_{z} \log\left(1 - D\left(G(z)\right)\right) \to 0$$

- There is no gradient signal to help the generator improve

Generator (G)'s loss function

- Instead of negating $J^{(D)}$, swap classes:

$$J^{(G)}\left(\theta^{(D)}, \theta^{(G)}\right) = -\frac{1}{2}\mathbb{E}_{x\sim p_{data}} \log\left(1 - D(x)\right) - \frac{1}{2}\mathbb{E}_z \log\left(D\left(G(z)\right)\right)$$

- The first term can be dropped, since $\theta^{(G)}$ does not influence it

$$J^{(G)}\left(\theta^{(D)}, \theta^{(G)}\right) = -\frac{1}{2}\mathbb{E}_z \log\left(D\left(G(z)\right)\right)$$

- Now when $D\left(G(z)\right) \to 0$, $-\frac{1}{2}\mathbb{E}_z \log\left(D\left(G(z)\right)\right) \to \infty$
- Gradient gets bigger when the discriminator gets better

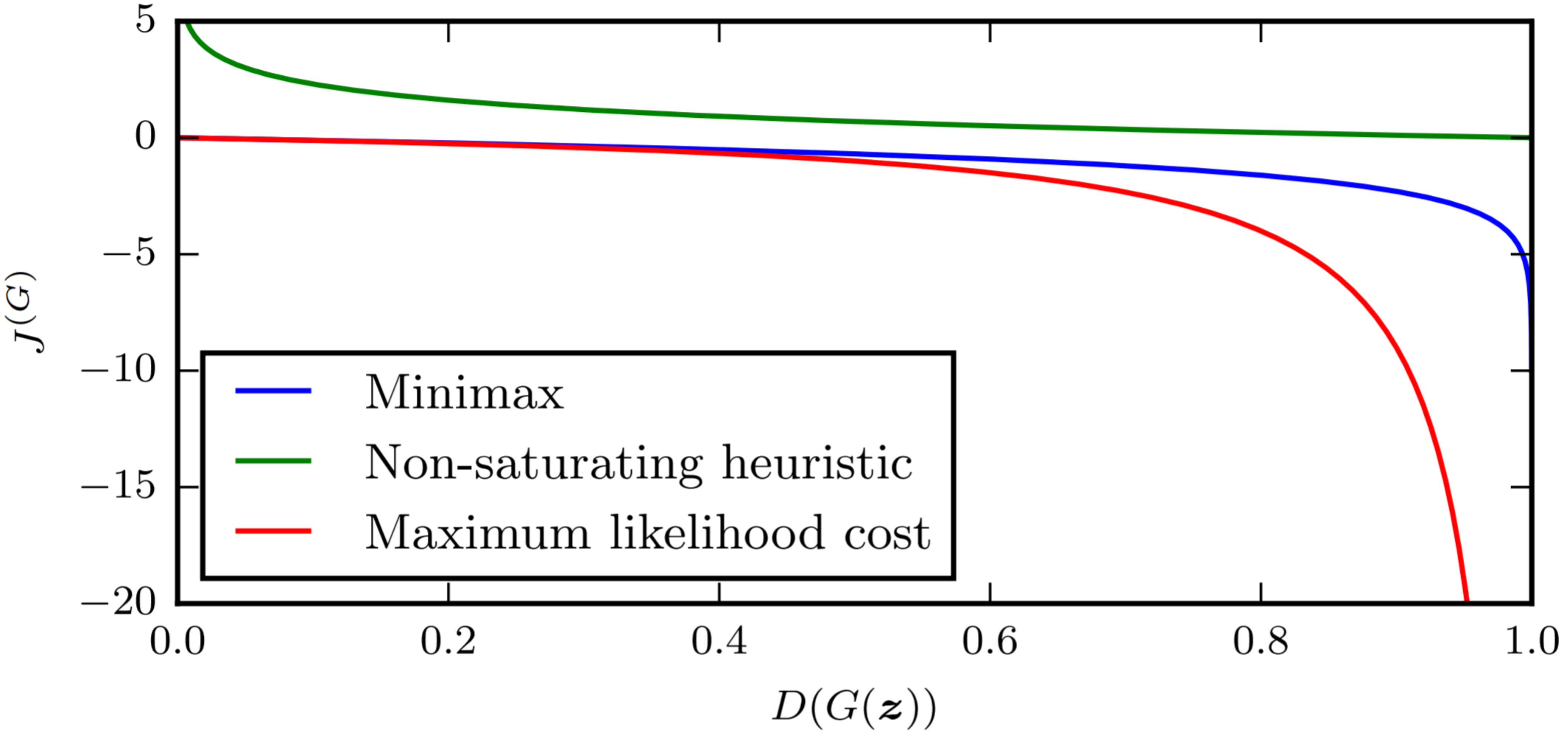
Making GANs approximate maximum likelihood

- Using a different choice of $J^{(G)}$, we can make GANs do maximum likelihood estimation
- Not typically used, but of theoretical interest

$$J^{(G)}\left(\theta^{(D)}, \theta^{(G)}\right) = -\frac{1}{2}\mathbb{E}_z \exp\left(\sigma^{-1}\left(D\left(G(z)\right)\right)\right)$$

- Where σ is the sigmoid function
- Can be shown this is equivalent to minimizing KL divergence between the data distribution and the model distribution under certain assumptions

Comparing G’s loss functions



Generator (G)’s loss function

- Because of the gradient, the original paper uses:

$$J^{(G)}\left(\theta^{(D)}, \theta^{(G)}\right) = -\frac{1}{2}\mathbb{E}_z \log\left(D\left(G(z)\right)\right)$$

- This function was later shown to give the same stationary point (under some assumptions) as $J^{(G)} = -J^{(D)}$

Other options in the loss

- Energy-based GAN (EBGAN) uses an "energy-based" discriminator function with a hinge loss (for example L2 loss of an autoencoder on real vs. fake examples):

$$J^{(D)}\left(\theta^{(D)}, \theta^{(G)}\right) = D(x) + \max(m - D\big(G(z)\big), 0)$$
$$J^{(G)}\left(\theta^{(D)}, \theta^{(G)}\right) = D(G(z))$$

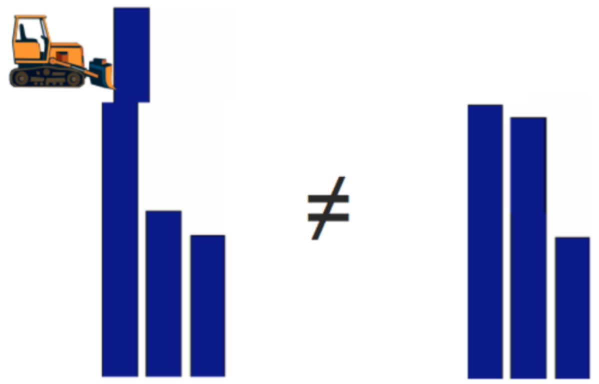
- Prove that this and many other choices mean that at a Nash equilibrium, $p_G(x) = p_{data}(x)$ almost everywhere
- The paper suggests several advantages, including more efficient training
- $J^{(G)}, J^{(D)}$ can both be modified (not arbitrarily): the game is what guides the learning

Different losses

- Choices of the loss function are further explored in Nowozin and colleagues f-GAN paper. They show a family of loss functions and how each corresponds to an f-divergence on the probability distributions we are trying to learn
- Arjovsky and colleages' Wasserstein GAN (WGAN) discusses the choice of divergence (and proposes using an approximation to the Earth Mover's distance)

WGAN

- If our data are on a low-dimensional manifold of a high dimensional space the model's manifold and the true data manifold can have a negligible intersection in practice
- KL divergence is undefined or infinite
- The loss function and gradients may not be continuous and well behaved
- The Earth Mover's Distance is well defined:
  - Minimum transportation cost for making one pile of dirt (pdf/pmf) look like the other

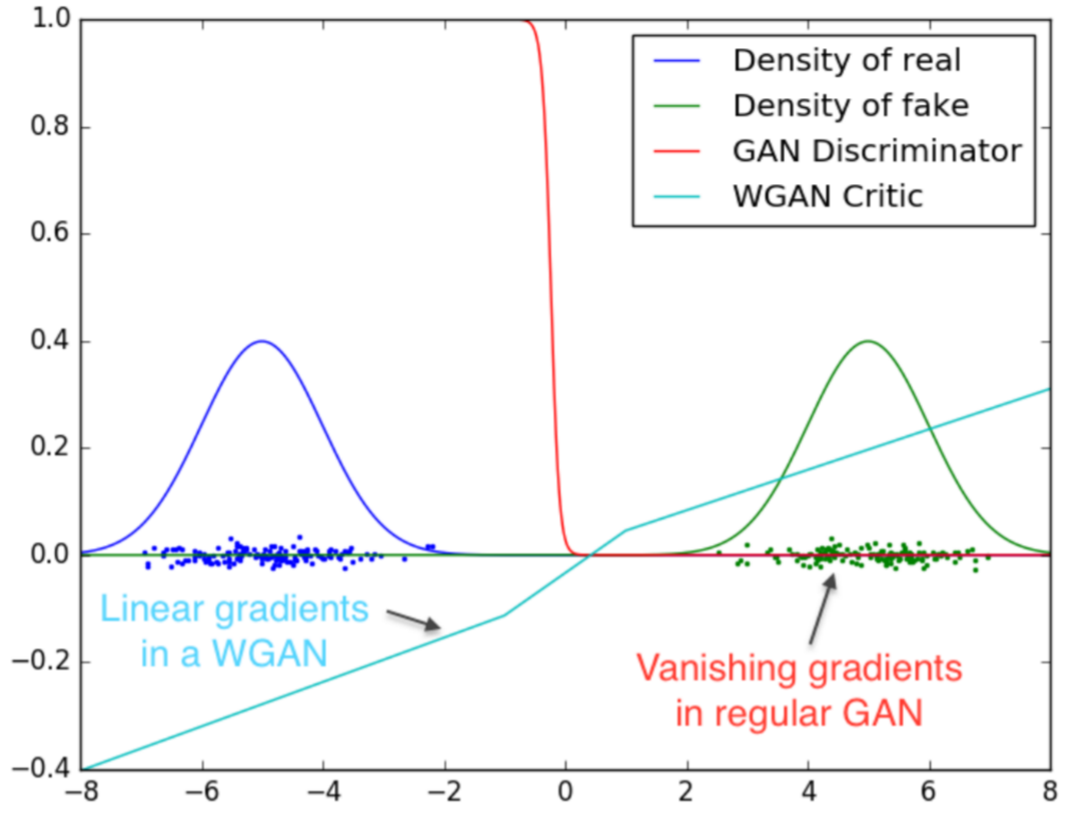


WGAN

$$J^{(D)}\left(\theta^{(D)}, \theta^{(G)}\right) = -\left[\mathbb{E}_{x\sim p_{data}} D(x) - \mathbb{E}_z D\left(G(z)\right)\right]$$
$$J^{(G)}\left(\theta^{(D)}, \theta^{(G)}\right) = -\mathbb{E}_z D\left(G(z)\right)$$

- Importantly, the discriminator is trained for many steps before the generator is updated
- Gradient-clipping is used in the discriminator to ensure $D(x)$ has the Lipschitz continuity required by the theory
- The authors argue that this solves many training issues, including mode collapse

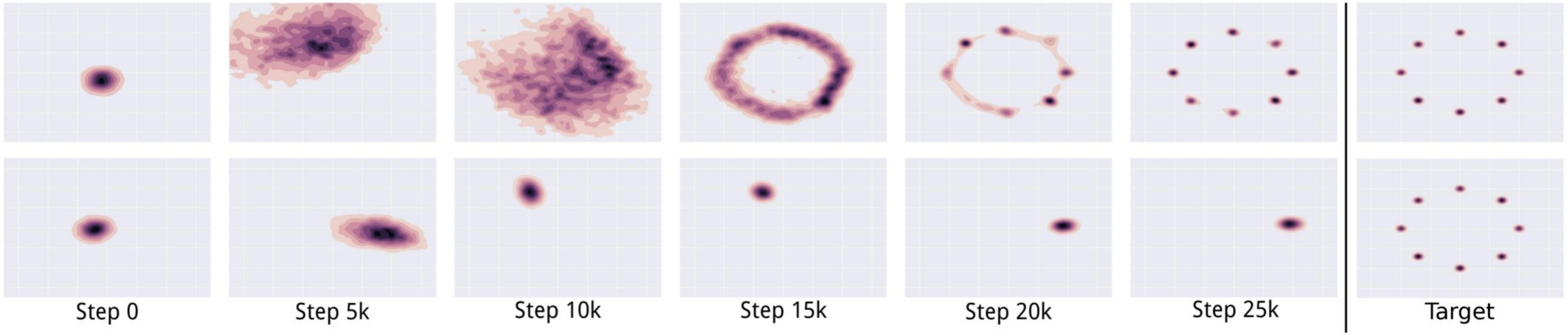
WGAN behavior



Loss function summary

- There are many choices of loss function
- Some choices have much better behavior during training
- Some choices will modify the latent space

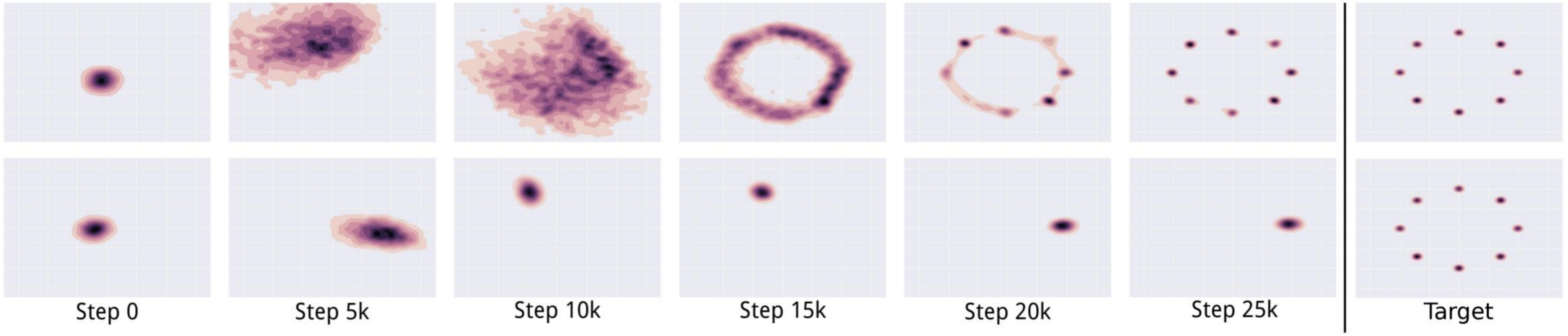
An optimization issue: Mode collapse



https://arxiv.org/pdf/1611.02163.pdf

- What prevents the generator from just picking the same example all the time?
- The top row finds all the modes, the bottom finds just one mode

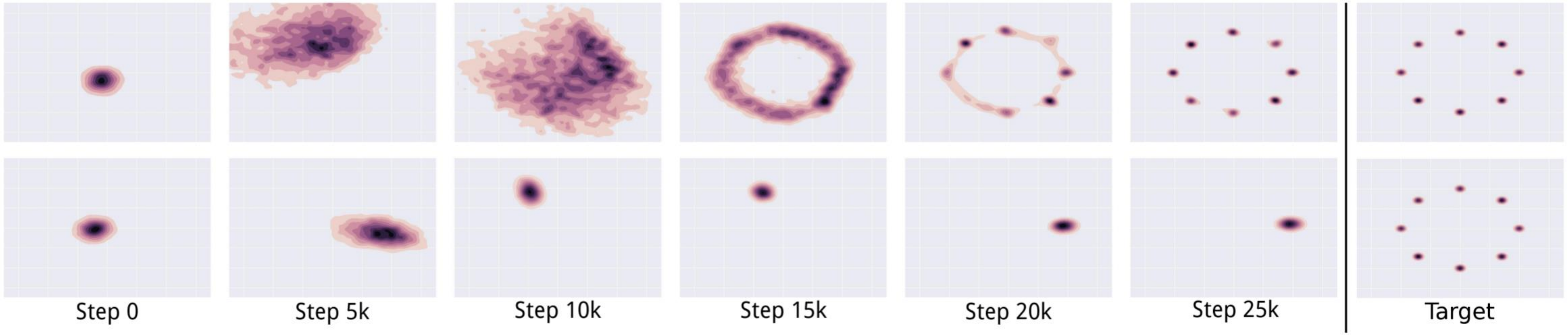
Mode collapse



https://arxiv.org/pdf/1611.02163.pdf

- Thought experiment: optimize the generator without changing the discriminator. What will happen?

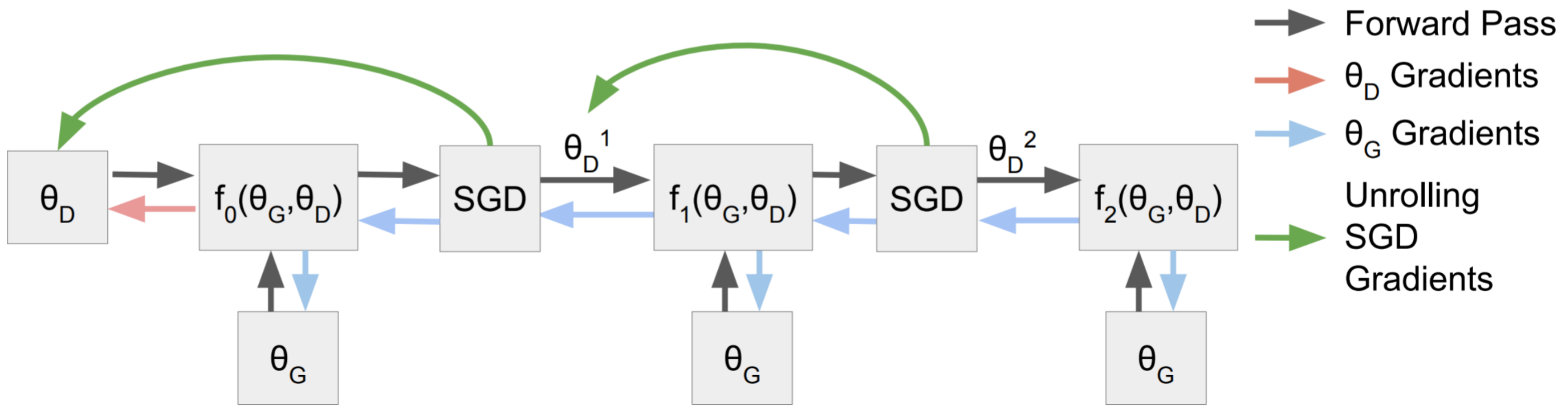
Mode collapse mitigation 1: minibatch features
(Salimans and colleagues, Improved Techniques for Training GANs)



https://arxiv.org/pdf/1611.02163.pdf

- Let the discriminator make a decision by comparing an example to a whole minibatch of fake/real examples
- Discriminator can now consider diversity

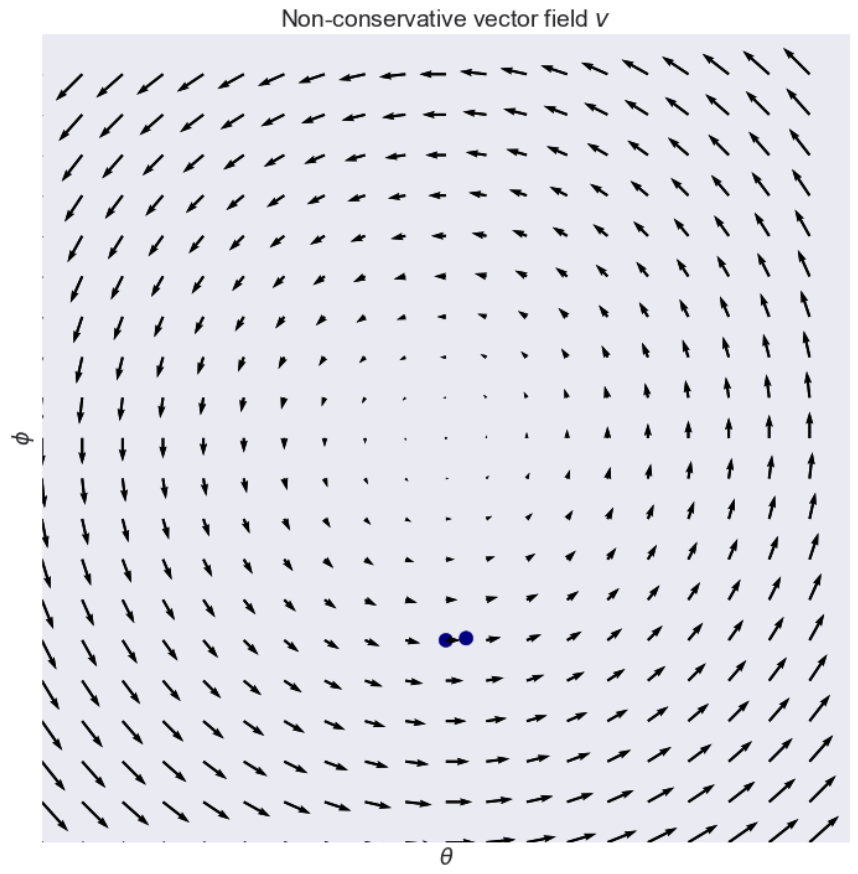
Mode collapse mitigation 2: unrolling (Metz and colleagues, Unrolled Generative Adversarial Networks)



https://arxiv.org/pdf/1611.02163.pdf

- Similar to Back-propagation through time, but now we back propagate through optimization steps
- We let the generator see where the discriminator would be after k steps before making its update
- The discriminator will react to the generator putting more mass somewhere by the putting less mass there: discourages the generator from concentrating mass

Does gradient descent make sense?



http://www.inference.vc/my-notes-on-the-numerics-of-gans/

- Does using gradient descent to find a Nash equilibrium make sense?
- This is not what gradient descent was designed for
- Each player moving down means the other moves up: can get stuck
- Classic example V(x, y) = -xy
- Mescheder and colleages, The numerics of GANs: Consensus optimization

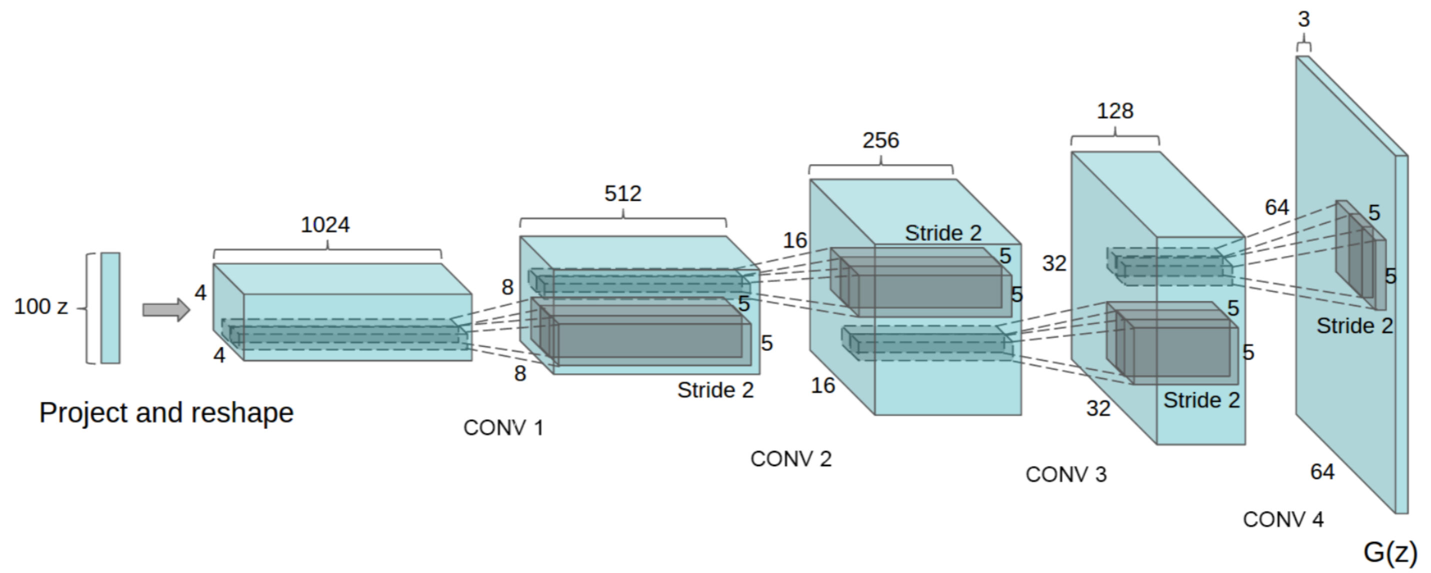
Story so far

- GANs provide a flexible framework for implicitly minimizing the divergence between the model and true probability distributions
- There are many choices of divergence
  - Some of these divergences are ill-defined for realistic settings
  - They can be poorly behaved
- Even when the divergence is well behaved, algorithms for finding a Nash equilibrium are not that good
  - Gradient descent is used, but the dynamics can prevent convergence
  - One interesting study: Li and colleagues, Towards Understanding the Dynamics of Generative Adversarial Networks
- Active research in training GANs: Lots of papers with "Towards" in the title

Evaluation

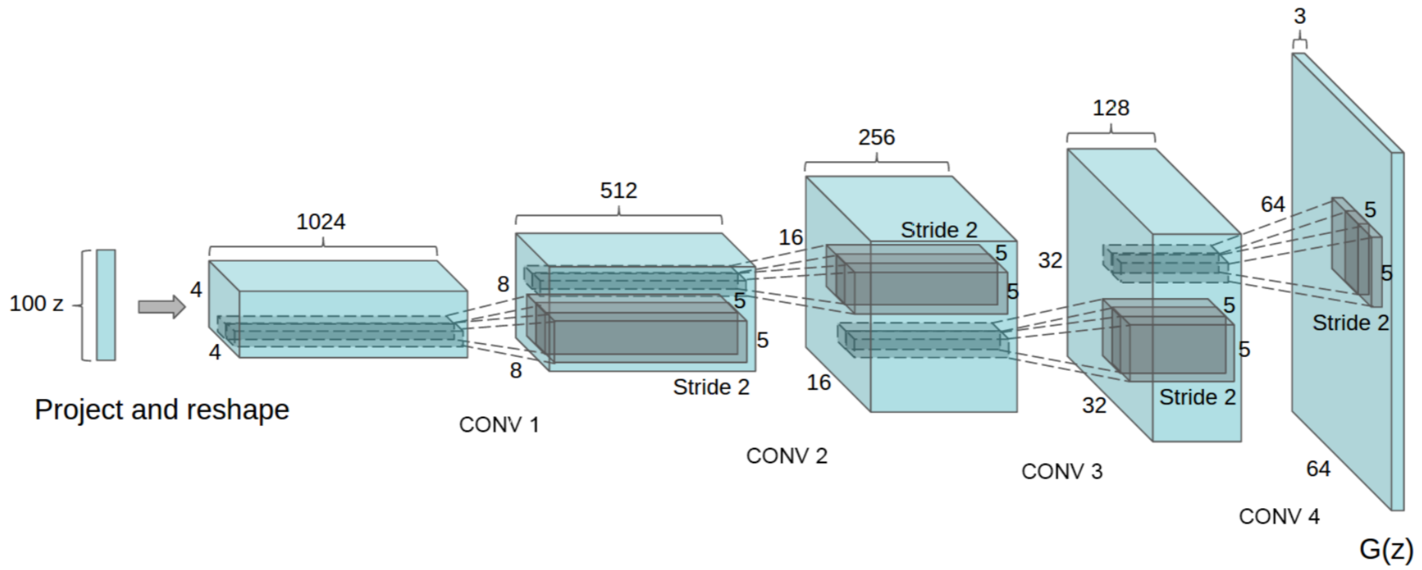
- Another issue with GANs is quantitative comparison
- There is no explicit likelihood to calculate
- Post hoc density estimation can be used, but is inaccurate
- Subjective evaluation by humans is currently the best method

Practical advice: DCGAN



- All-convolutional network: no pooling layers, strided transpose convolution
- ADAM optimization
- Batch normalization
    - Not in last layer of G, not in first layer of D: learn mean/scale of data
    - The two minibatches for the discriminator are normalized separately

Practical advice: DCGAN



- Why does this work? Purely empirical. They tried a bunch of architectures
- This architecture seems to somehow constrain the model distribution so that many of the training problems are mitigated

Practical advice: One-sided label smoothing

- If using the original

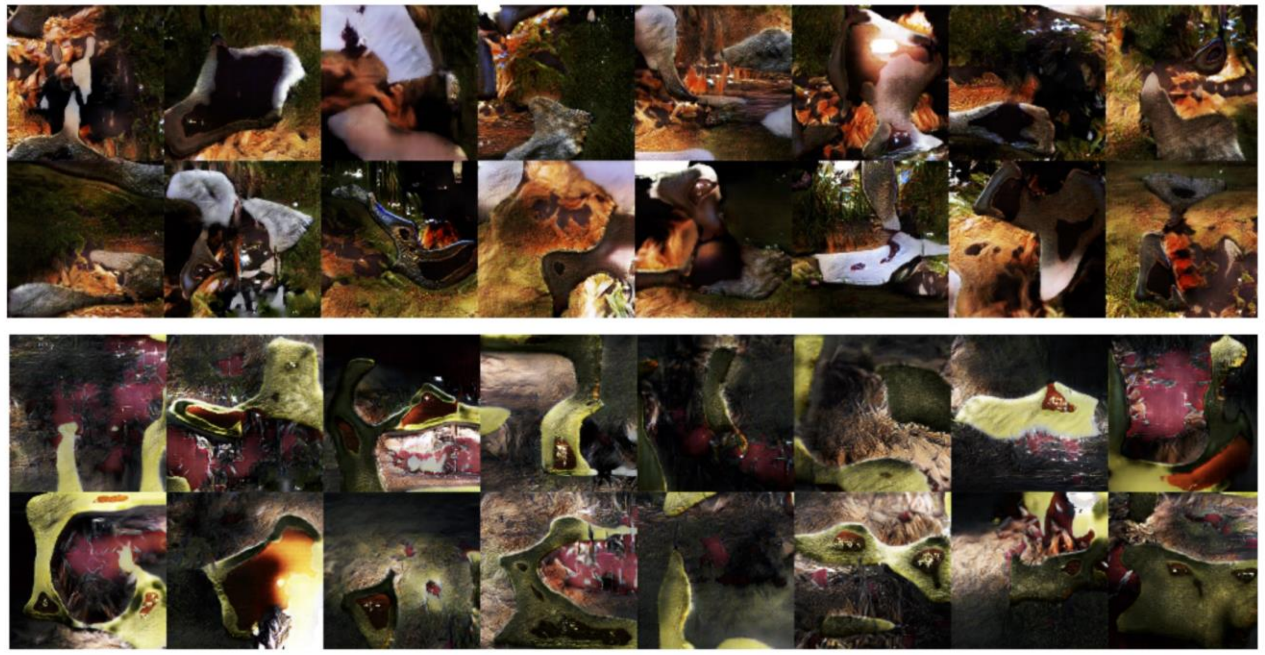
$$J^{(D)}\left(\theta^{(D)}, \theta^{(G)}\right) = -\frac{1}{2}\mathbb{E}_{x \sim p_{data}} \log D(x) - \frac{1}{2}\mathbb{E}_{z} \log\left(1 - D\left(G(z)\right)\right)$$

- It can be helpful to decrease the confidence of the discriminator by setting the target of the real examples to 0.9 e.g. instead of 1 (but keep the target of the model at exactly 0)
- Keeps the logits at smaller values and mitigates “extrapolation” to new data (overfitting)

Practical advice: add noise

- For a similar reason, it can be useful to add noise to the data
- This helps prevent discriminator overfitting, and also helps with the problem of non-overlapping support between the model and data distributions

Practical advice: virtual batch normalization



https://arxiv.org/pdf/1701.00160.pdf

- Batch normalization causes generated samples to become correlated
- Use a reference batch to do batch normalization (use the statistics from the reference)
- Or use a reference batch combined with the current batch (compute statistics from the combined batch)
- Batch renormalization is another option

Practical advice: use labels if available

- GANs can be used in a supervised or semi-supervised setting
- One way to do this is to give both the discriminator and the generator the label, making them class conditional
- Another way to do this is to change the discriminator to predict n + 1 classes, where a class is added for fake data
- Using labels dramatically improves the sample quality

Relationship to Reinforcement Learning

- We’ll see reinforcement learning later in the course
- Similar to GANs in the sense that the actions taken by a player are rewarded, and the reward function governs learning
- Squinting our eyes, there are similarities
- But in GANs:
    - The reward function changes in response to changes in the generator (there are two players responding to each other)
    - The generator gets to observe gradients of the reward, not just the reward
- GANs can be formally related to inverse reinforcement learning

Summary

- The GAN framework is a powerful way to do unsupervised learning
- The samples from the GAN model are state of the art (FVBN models are competitive though)
- Training GANs is very difficult for fundamental reasons, and this is an area of active research
- Very popular with many variants. Some add encoders (BiGAN), make the latent code more interpretable (InfoGAN), and there are many others
    - https://github.com/hindupuravinash/the-gan-zoo