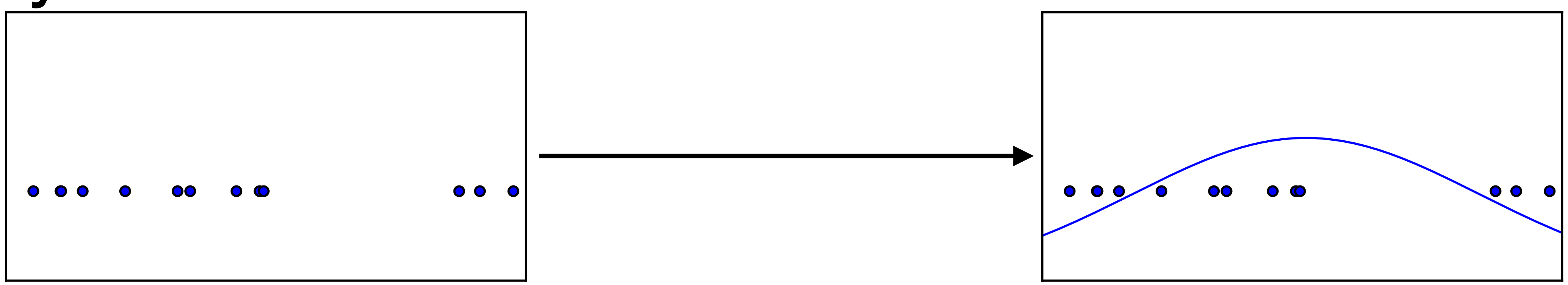
Generative Adversarial Networks (GANs)

Based on slides from Ian Goodfellow's NIPS 2016 tutorial

Generative Modeling

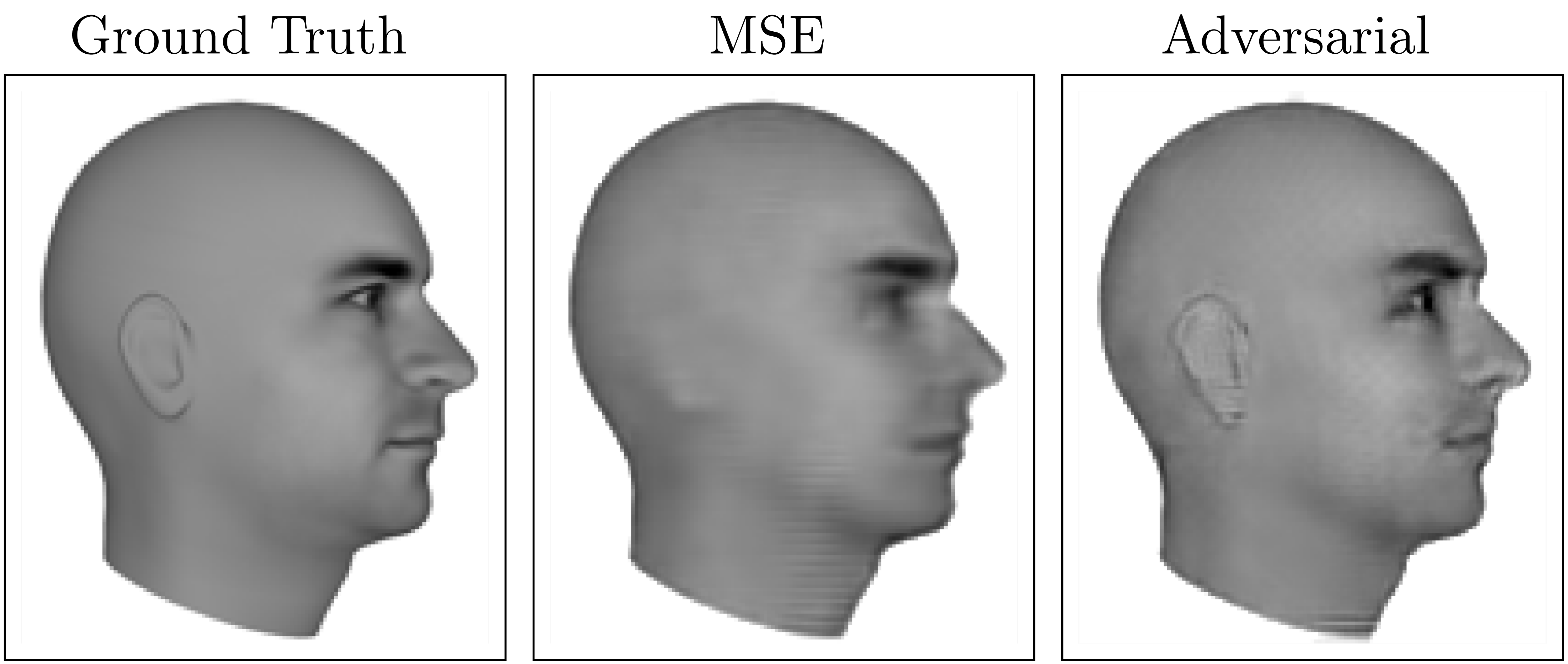
- Density estimation



- Sample generation

Training examples Model samples

Next Video Frame Prediction

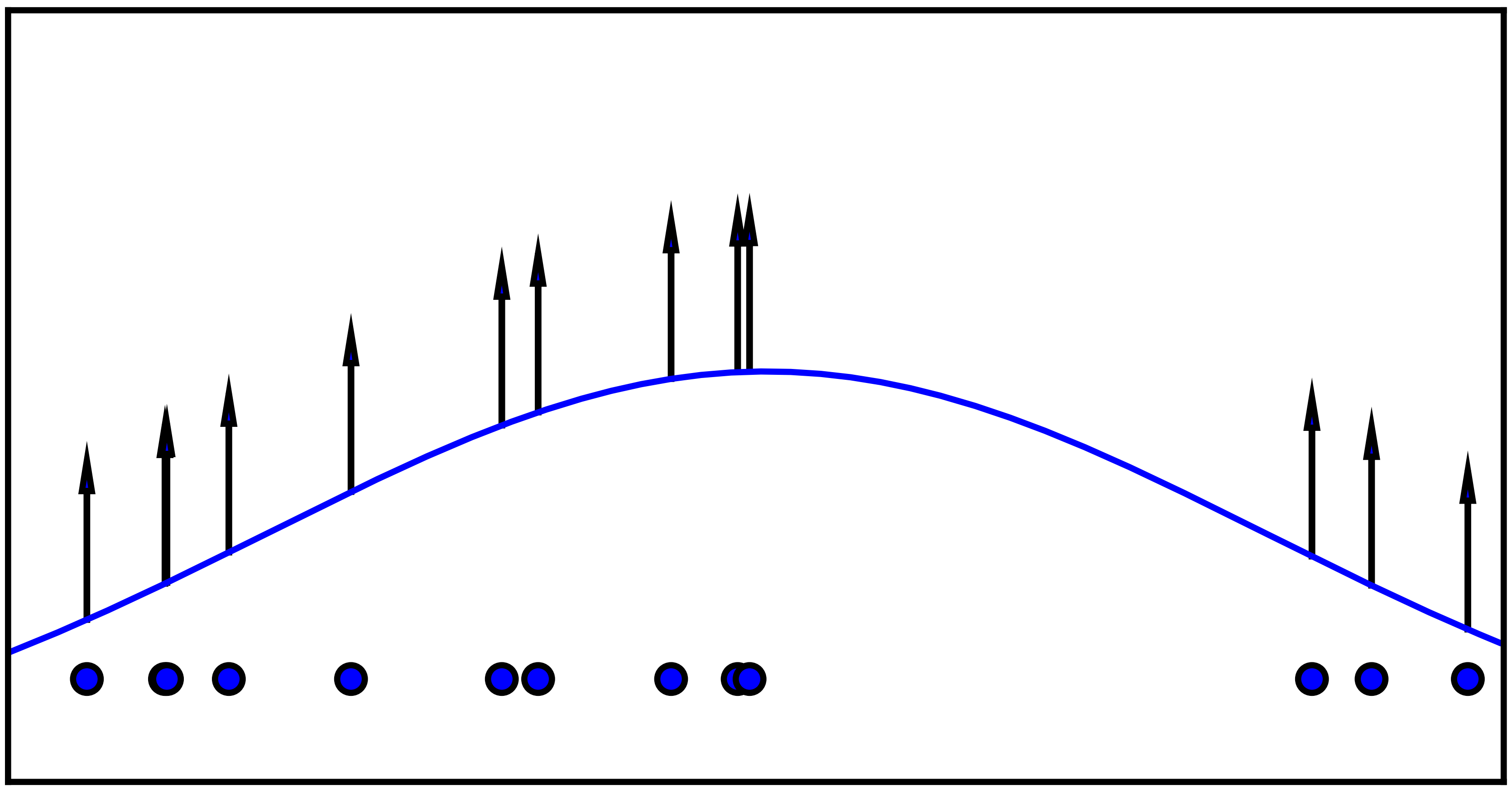


(Lotter et al 2016)

Image to Image Translation

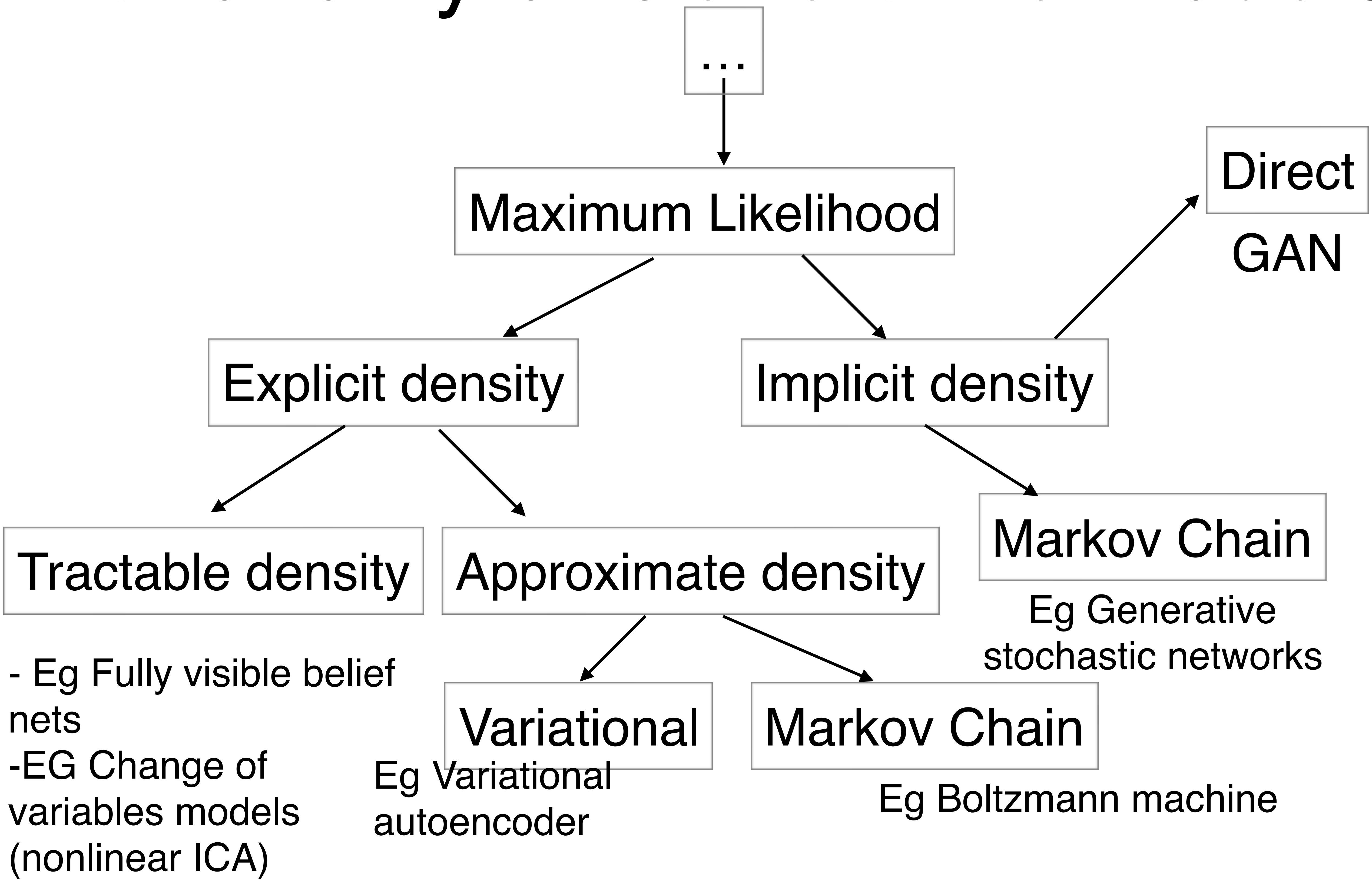
(Isola et al 2016)

Maximum Likelihood



$$\boldsymbol{\theta}^* = \arg\max_{\boldsymbol{\theta}} \mathbb{E}_{x \sim p_{\text{data}}} \log p_{\text{model}}(\boldsymbol{x} \mid \boldsymbol{\theta})$$

Taxonomy of Generative Models



Fully Visible Belief Nets

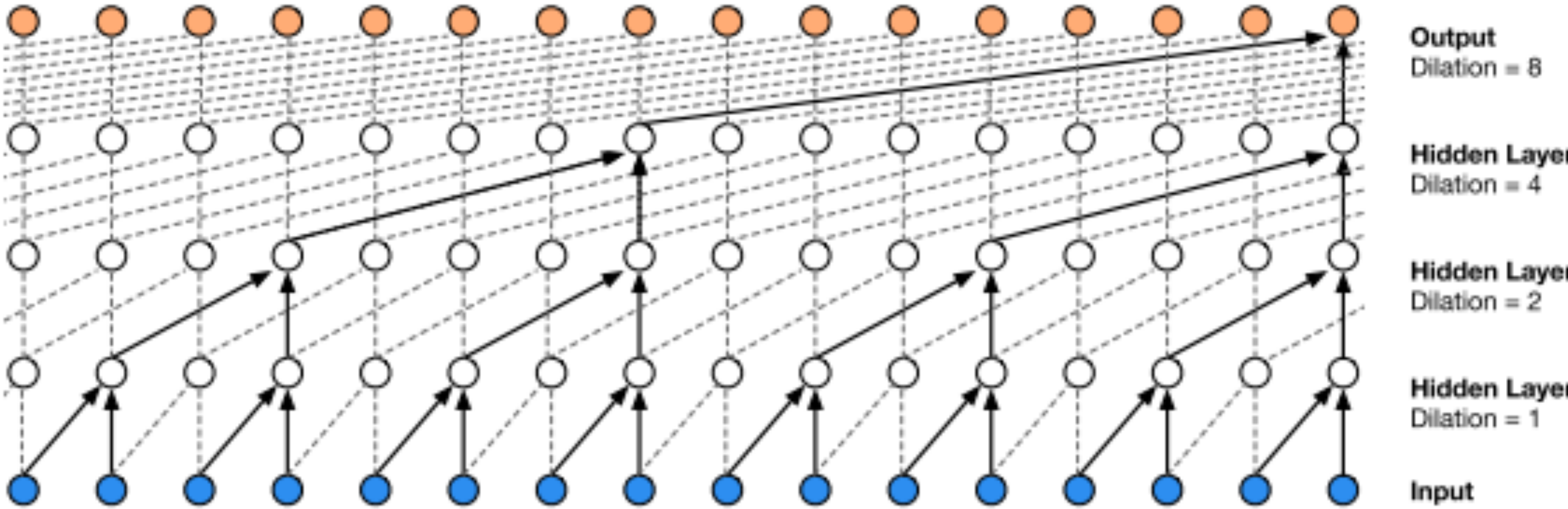
- Explicit formula based on chain rule: (Frey et al, 1996)

$$p_{\text{model}}(\boldsymbol{x}) = p_{\text{model}}(x_1) \prod_{i=2}^{n} p_{\text{model}}(x_i \mid x_1, \ldots, x_{i-1})$$

- Disadvantages:
  - O(n) sample generation cost
  - Generation not controlled by a latent code

PixelCNN elephants
(van den Ord et al 2016)

WaveNet



Amazing quality
Sample generation slow

Two minutes to synthesize
one second of audio

Change of Variables

$$y = g(x) \Rightarrow p_x(\boldsymbol{x}) = p_y(g(\boldsymbol{x})) \left| \det \left(\frac{\partial g(\boldsymbol{x})}{\partial \boldsymbol{x}} \right) \right|$$

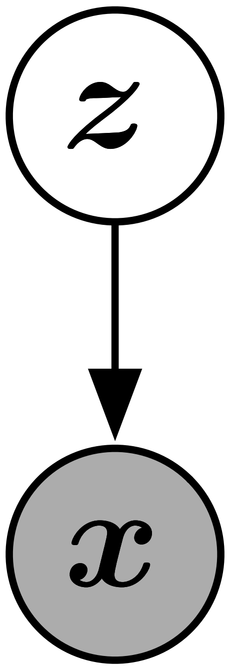
e.g. Nonlinear ICA (Hyvärinen 1999)

64x64 ImageNet Samples
Real NVP (Dinh et al 2016)

Disadvantages:
- Transformation must be invertible
- Latent dimension must match visible dimension

Variational Autoencoder

(Kingma and Welling 2013, Rezende et al 2014)



$$\log p(\boldsymbol{x}) \geq \log p(\boldsymbol{x}) - D_{\mathrm{KL}}\left(q(\boldsymbol{z}) \| p(\boldsymbol{z} \mid \boldsymbol{x})\right)$$
$$= \mathbb{E}_{\boldsymbol{z} \sim q} \log p(\boldsymbol{x}, \boldsymbol{z}) + H(q)$$

CIFAR-10 samples
(Kingma et al 2016)

Disadvantages:
-Not asymptotically consistent unless *q* is perfect
-Samples tend to have lower quality

Boltzmann Machines

$$p(\boldsymbol{x}) = \frac{1}{Z} \exp\left(-E(\boldsymbol{x}, \boldsymbol{z})\right)$$

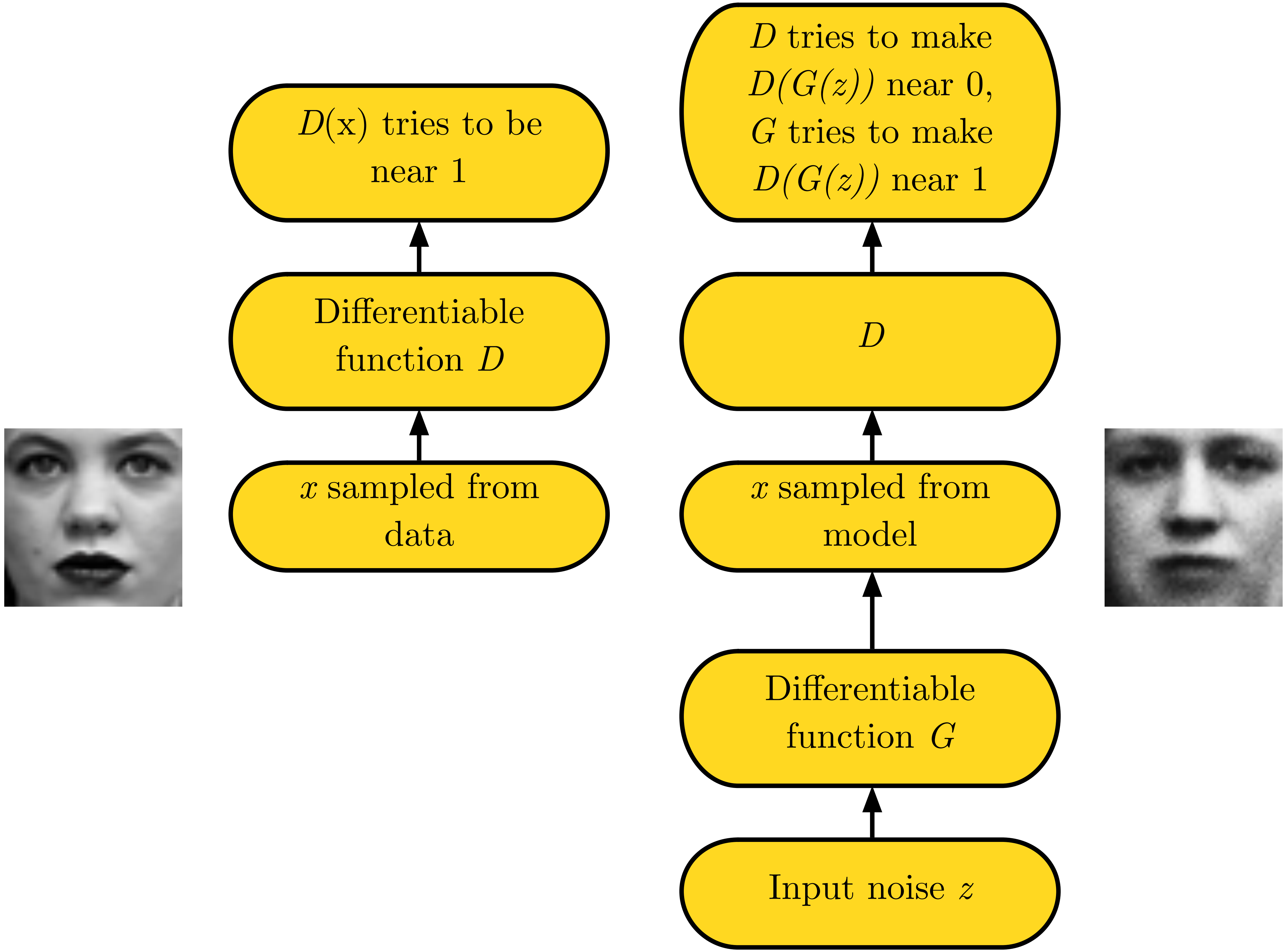
$$Z = \sum_{\boldsymbol{x}} \sum_{\boldsymbol{z}} \exp\left(-E(\boldsymbol{x}, \boldsymbol{z})\right)$$

- Partition function is intractable
- May be estimated with Markov chain methods
- Generating samples requires Markov chains too

GANs

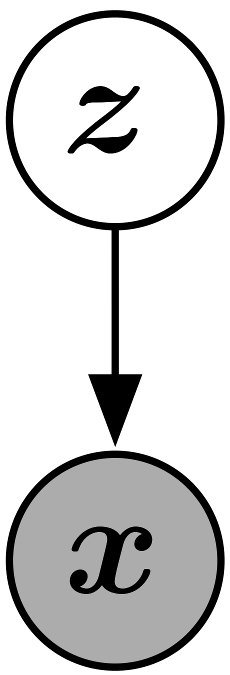
- Use a latent code
- Asymptotically consistent (unlike variational methods)
- No Markov chains needed
- Often regarded as producing the best samples
  - No good way to quantify this

Adversarial Nets Framework



Generator Network

$$\boldsymbol{x} = G(\boldsymbol{z}; \boldsymbol{\theta}^{(G)})$$



-Must be differentiable

- No invertibility requirement
- Trainable for any size of ***z***
- Some guarantees require ***z*** **to have higher dimension than** *x*
- **Can make *x* conditionally Gaussian given** *z* **but need not do so**

Training Procedure

- Use SGD-style updates on two minibatches simultaneously:
    - A minibatch of training examples
    - A minibatch of generated samples
- Optional: run *k* steps of one player for every step of the other player.

Minimax Game

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

$$J^{(G)} = -J^{(D)}$$

-Equilibrium is a saddle point of the discriminator loss

-Resembles Jensen-Shannon divergence:
JSD(P,Q) = 0.5 DKL(P,M) + 0.5 DKL(Q,M)
where M=0.5P + 0.5Q

-Generator minimizes the log-probability of the discriminator being correct

Exercise 1

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

$$J^{(G)} = -J^{(D)}$$

- What is the solution to $D(\boldsymbol{x})$ in terms of p_{data} and $p_{\text{generator}}$?
- What assumptions are needed to obtain this solution?

Solution

- Assume both densities are nonzero everywhere
  - If not, some input values *x* are never trained, so some values of *D*(*x*) have undetermined behavior.
- Solve for where the functional derivatives are zero:

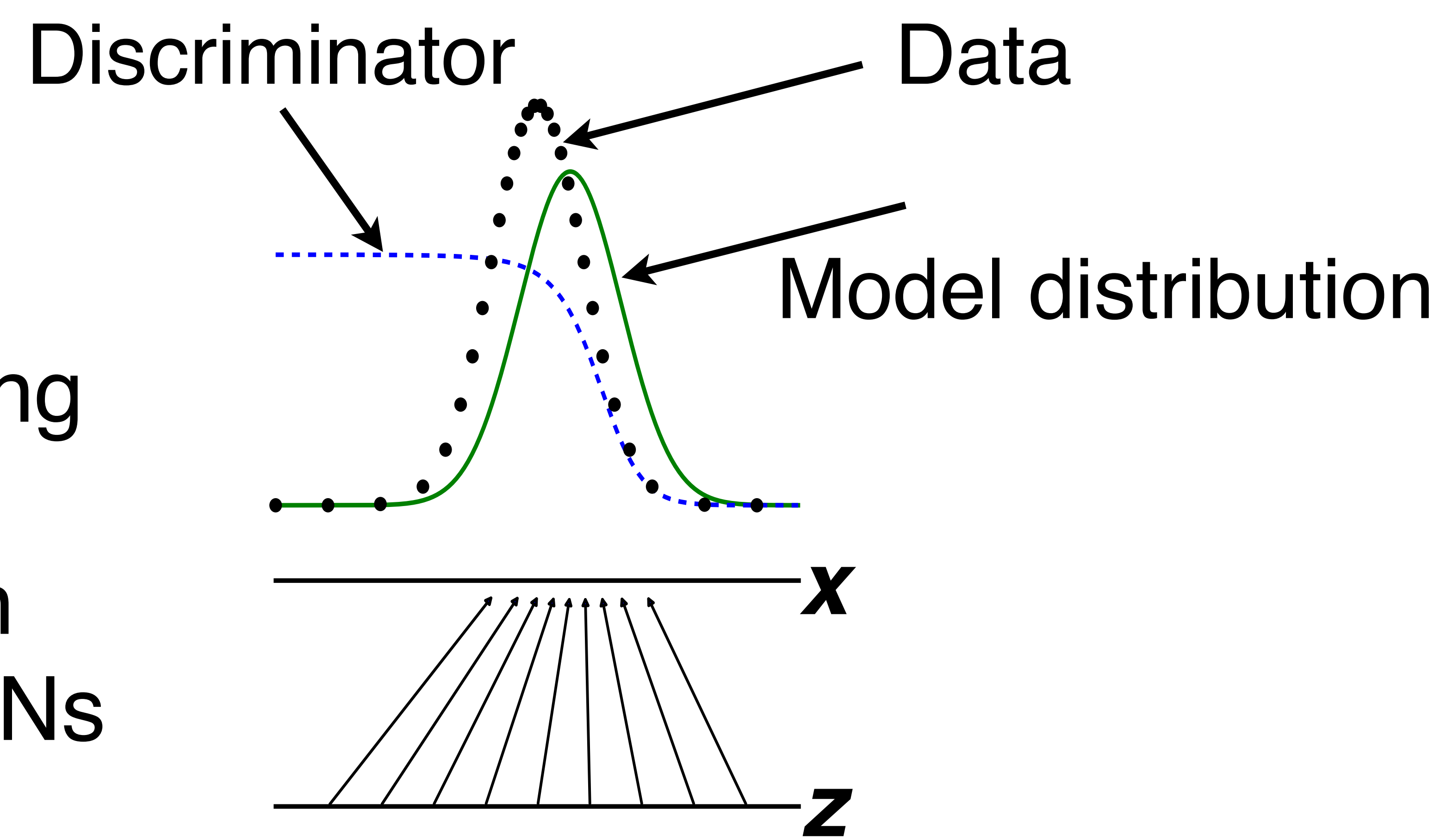
$$\frac{\delta}{\delta D(\boldsymbol{x})} J^{(D)} = 0$$

Discriminator Strategy

Optimal $D(\boldsymbol{x})$ for any $p_{\text{data}}(\boldsymbol{x})$ and $p_{\text{model}}(\boldsymbol{x})$ is always

$$D(x) = \frac{p_{\text{data}}(x)}{p_{\text{data}}(x) + p_{\text{model}}(x)}$$

Estimating this ratio
using supervised learning
is
the key approximation
mechanism used by GANs



Non-Saturating Game

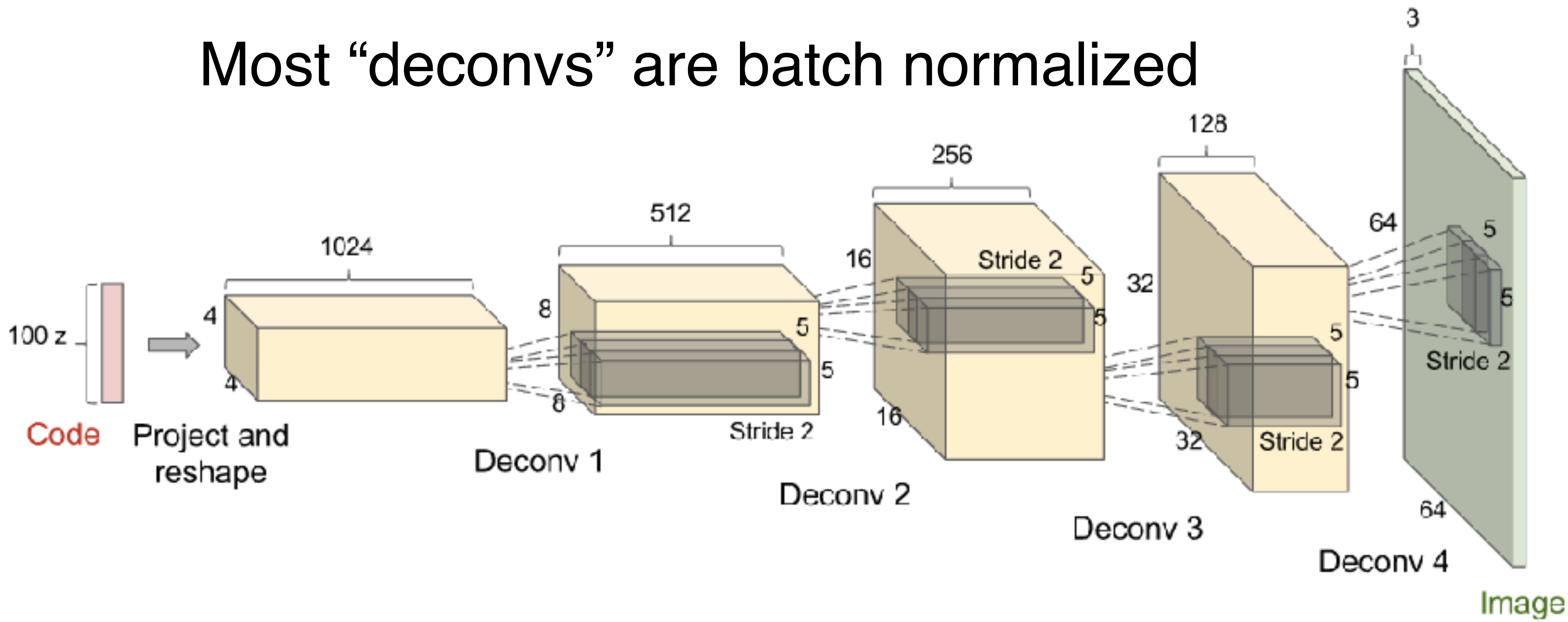
$$J^{(D)} = -\frac{1}{2}\mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}} \log D(\boldsymbol{x}) - \frac{1}{2}\mathbb{E}_{\boldsymbol{z}} \log\left(1 - D\left(G(\boldsymbol{z})\right)\right)$$

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

-Equilibrium no longer describable with a single loss
-Generator maximizes the log-probability of the discriminator being mistaken
-Heuristically motivated; generator can still learn even when discriminator successfully rejects all generator samples

DCGAN Architecture

Most “deconvs” are batch normalized



(Radford et al 2015)

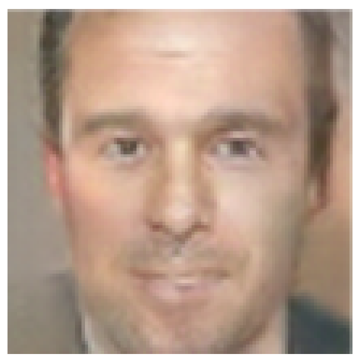
DCGANs for LSUN Bedrooms

(Radford et al 2015)

Vector Space Arithmetic

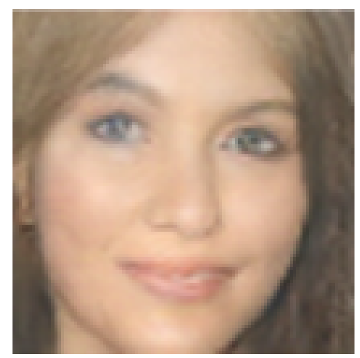
Man
with glasses

-



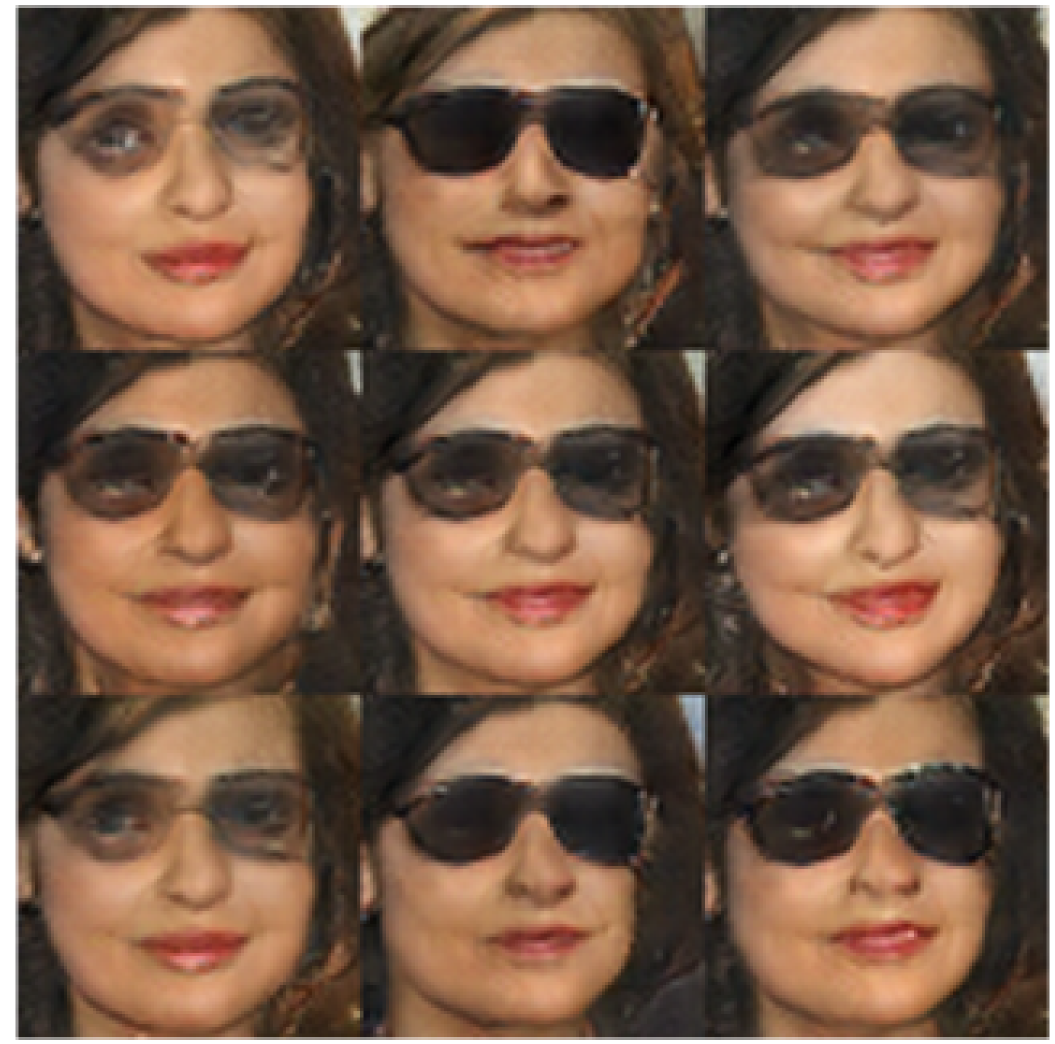
Man

+



Woman

=

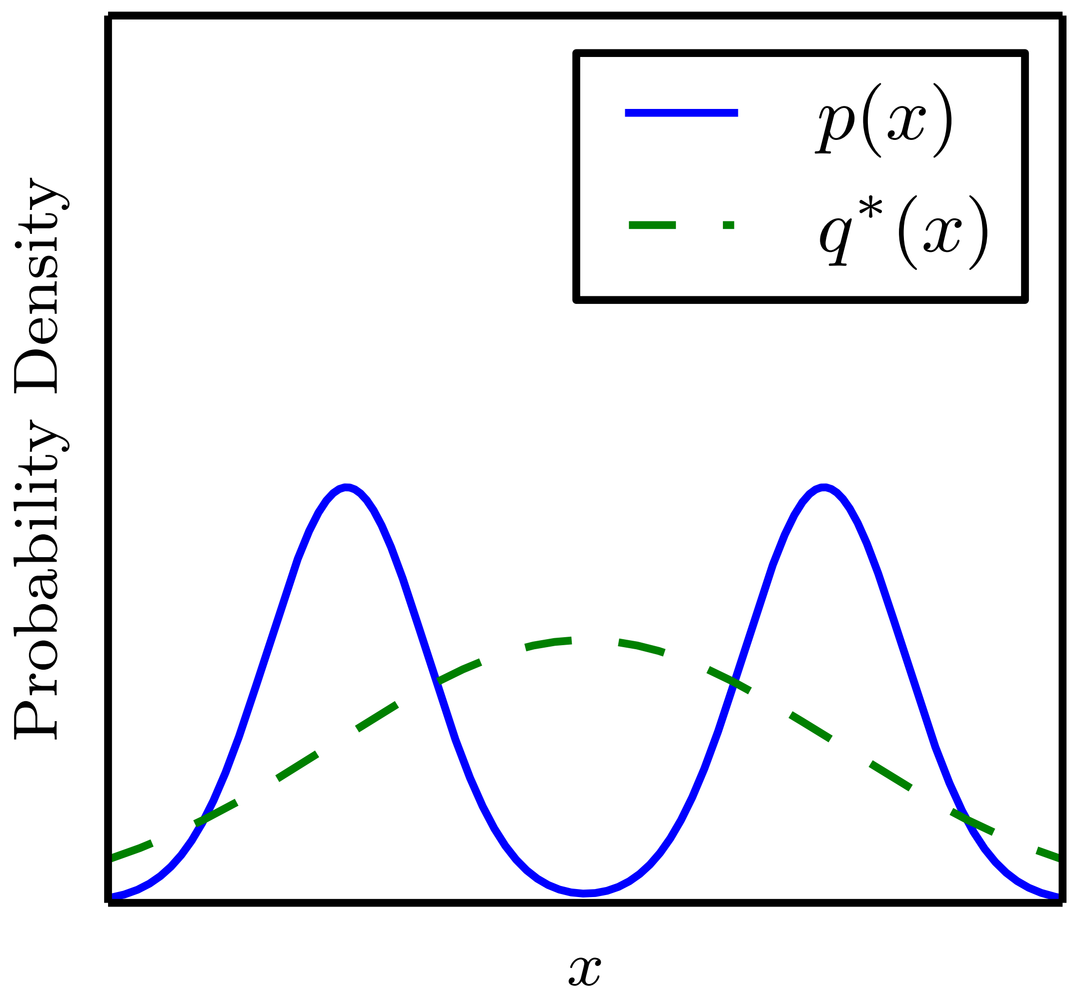


Woman with Glasses

(Radford et al, 2015)

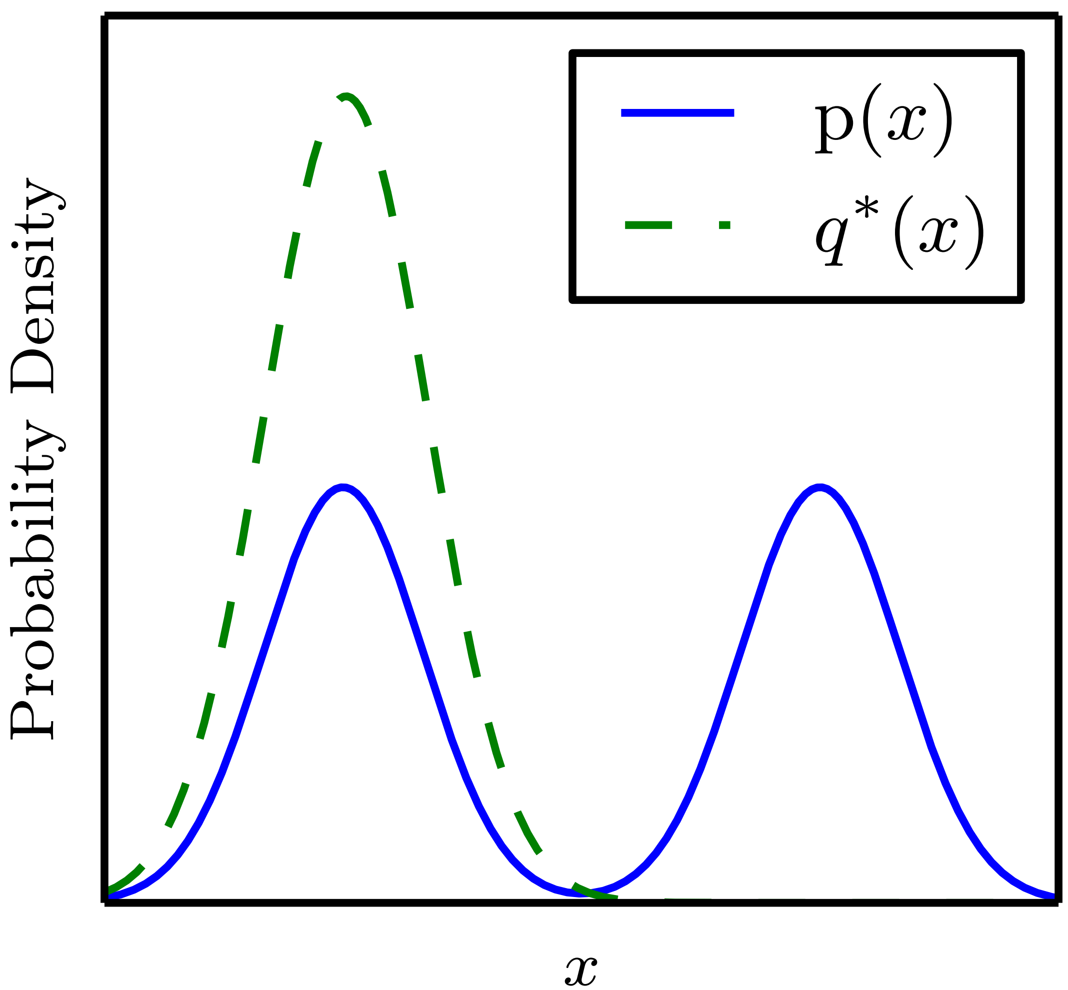
Is the divergence important?

$$q^* = \text{argmin}_q D_{\text{KL}}(p\|q)$$



Maximum likelihood

$$q^* = \text{argmin}_q D_{\text{KL}}(q\|p)$$



Reverse KL

(Goodfellow et al 2016)

Modifying GANs to do Maximum Likelihood

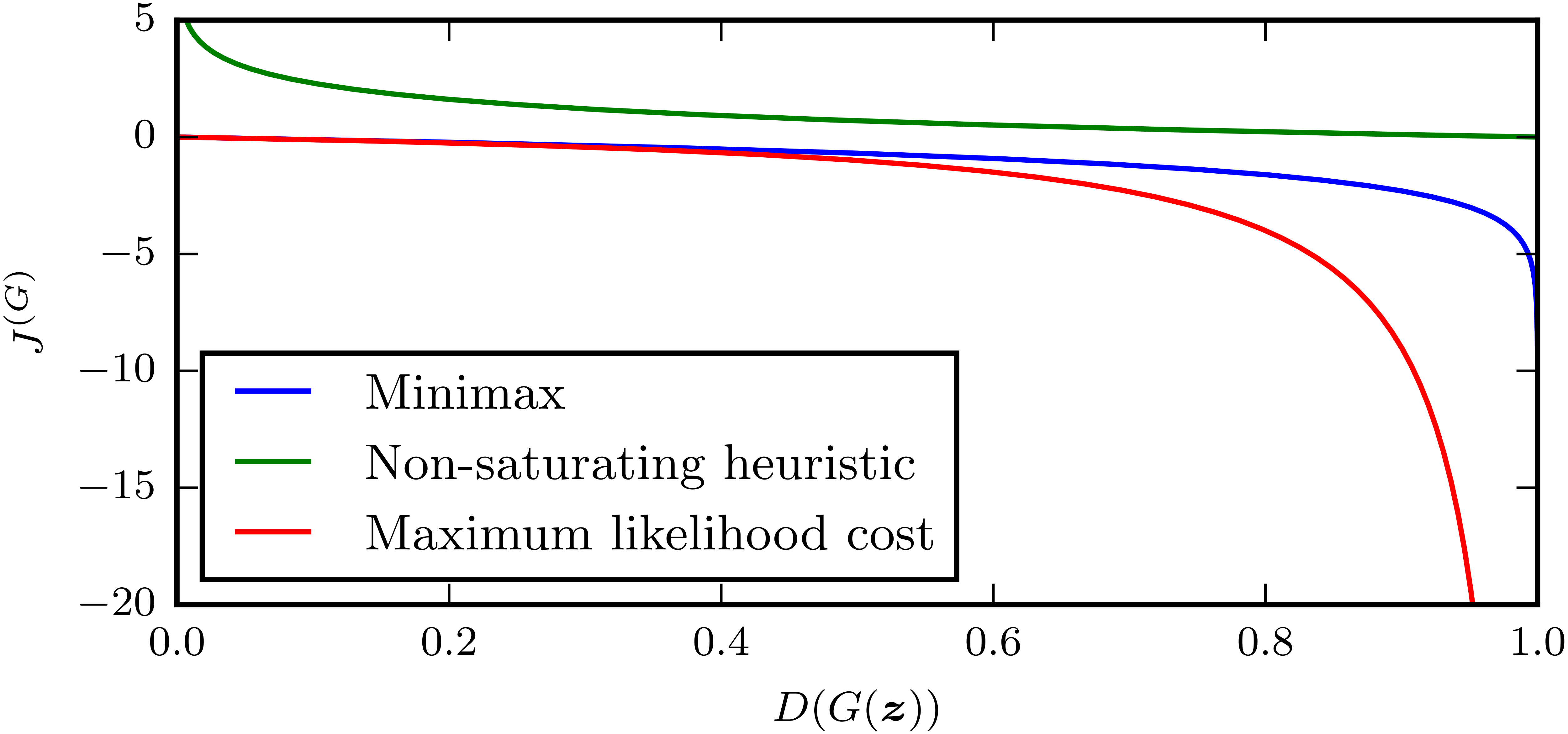
$$J^{(D)} = -\frac{1}{2}\mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}} \log D(\boldsymbol{x}) - \frac{1}{2}\mathbb{E}_{\boldsymbol{z}} \log\left(1 - D\left(G(\boldsymbol{z})\right)\right)$$

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

When discriminator is optimal, the generator gradient matches that of maximum likelihood

(“On Distinguishability Criteria for Estimating Generative Models”, Goodfellow 2014, pg 5)

Comparison of Generator Losses



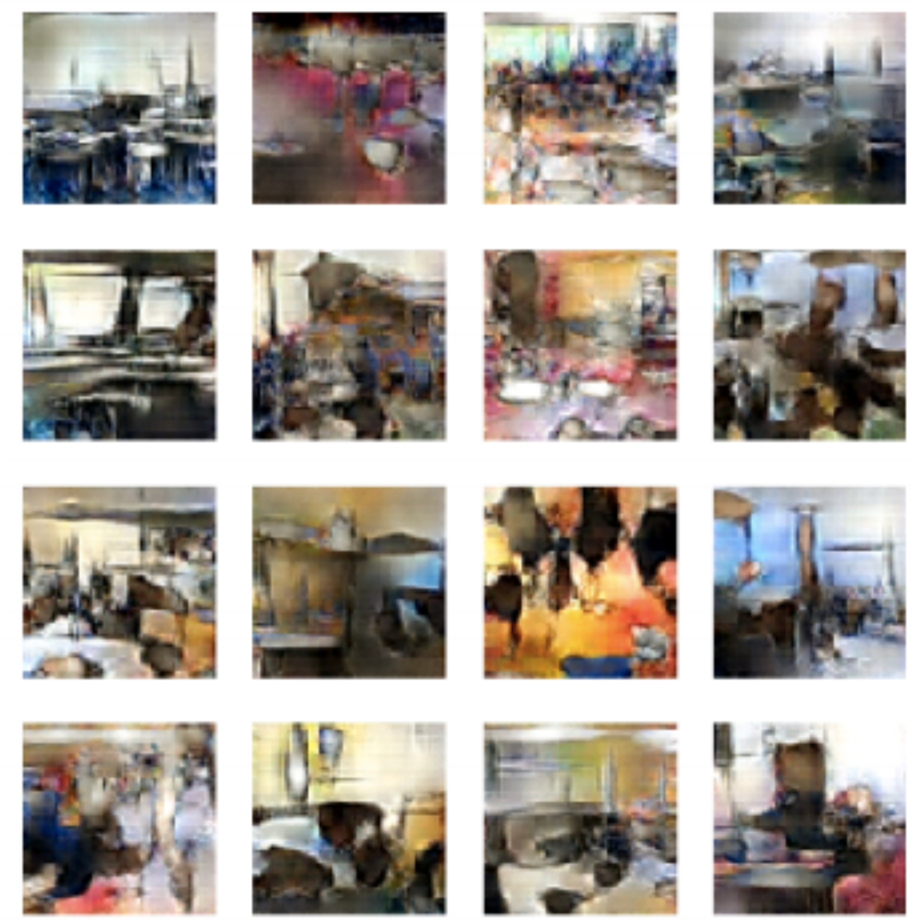
(Goodfellow 2014)

Loss does not seem to explain why GAN samples are sharp

KL

Reverse KL

(Nowozin et al 2016)



KL samples from LSUN

Takeaway: the approximation strategy matters more than the loss

Labels improve subjective sample quality

- Learning a conditional model *p(y|x)* often gives much better samples from all classes than learning *p*(*x*) does (Denton et al 2015)

- Even just learning *p(x,y)* makes samples from *p(x)* look much better to a human observer (Salimans et al 2016)

- Note: this defines three categories of models (no labels, trained with labels, generating condition on labels) that should not be compared directly to each other

One-sided label smoothing

- Default discriminator cost:

```
cross_entropy(1., discriminator(data))
+ cross_entropy(0., discriminator(samples))
```

- One-sided label smoothed cost (Salimans et al 2016):

```
cross_entropy(.9, discriminator(data))
+ cross_entropy(0., discriminator(samples))
```

Do not smooth negative labels

```
cross_entropy(1.-alpha, discriminator(data))
+ cross_entropy(beta, discriminator(samples))
```

Reinforces current generator behavior

$$D(\boldsymbol{x}) = \frac{(1-\alpha)p_{\text{data}}(\boldsymbol{x}) + \beta p_{\text{model}}(\boldsymbol{x})}{p_{\text{data}}(\boldsymbol{x}) + p_{\text{model}}(\boldsymbol{x})}$$

Benefits of label smoothing

- Good regularizer (Szegedy et al 2015)
- Does not reduce classification accuracy, only confidence
- Benefits specific to GANs:
    - Prevents discriminator from giving very large gradient signal to generator
    - Prevents extrapolating to encourage extreme samples

Batch Norm

- Given inputs $\boldsymbol{X}=\{\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, .., \boldsymbol{x}^{(m)}\}$
- **Compute mean and standard deviation of features of *X***
- **Normalize features (subtract mean, divide by standard deviation)**
- **Normalization operation is part of the graph**
    - **Backpropagation computes the gradient through the normalization**
    - **This avoids wasting time repeatedly learning to undo the normalization**

Batch norm in G can cause strong intra-batch correlation

Reference Batch Norm

- Fix a *reference batch* $R=\{\boldsymbol{r}^{(1)}, \boldsymbol{r}^{(2)}, .., \boldsymbol{r}^{(m)}\}$
- Given new inputs $\boldsymbol{X}=\{\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, .., \boldsymbol{x}^{(m)}\}$
- **Compute mean and standard deviation of features of *R***
  - Note that though *R* does not change, the feature values change when the parameters change
- Normalize the features of *X* using the mean and standard deviation from *R*
- **Every $\boldsymbol{x}^{(i)}$ is always treated the same, regardless of which other examples appear in the minibatch**

Virtual Batch Norm

- Reference batch norm can overfit to the reference batch. A partial solution is *virtual batch norm*
- Fix a *reference batch* $R=\{\boldsymbol{r}^{(1)}, \boldsymbol{r}^{(2)}, .., \boldsymbol{r}^{(m)}\}$
- Given new inputs $\boldsymbol{X}=\{\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, .., \boldsymbol{x}^{(m)}\}$
- **For each $\boldsymbol{x}^{(i)}$ in *X*:**
  - **Construct a *virtual batch V* containing both $\boldsymbol{x}^{(i)}$ and all of *R***
  - **Compute mean and standard deviation of features of V**
  - Normalize the features of $\boldsymbol{x}^{(i)}$ using the mean and standard deviation from *V*

Balancing *G* and *D*

- Usually the discriminator “wins”
- This is a good thing—the theoretical justifications are based on assuming *D* is perfect
- Usually *D* is bigger and deeper than *G*
- Sometimes run *D* more often than *G*. Mixed results.
- Do not try to limit *D* to avoid making it “too smart”
  - Use non-saturating cost
  - Use label smoothing

Non-convergence

- Optimization algorithms often approach a saddle point or local minimum rather than a global minimum
- Game solving algorithms may not approach an equilibrium at all

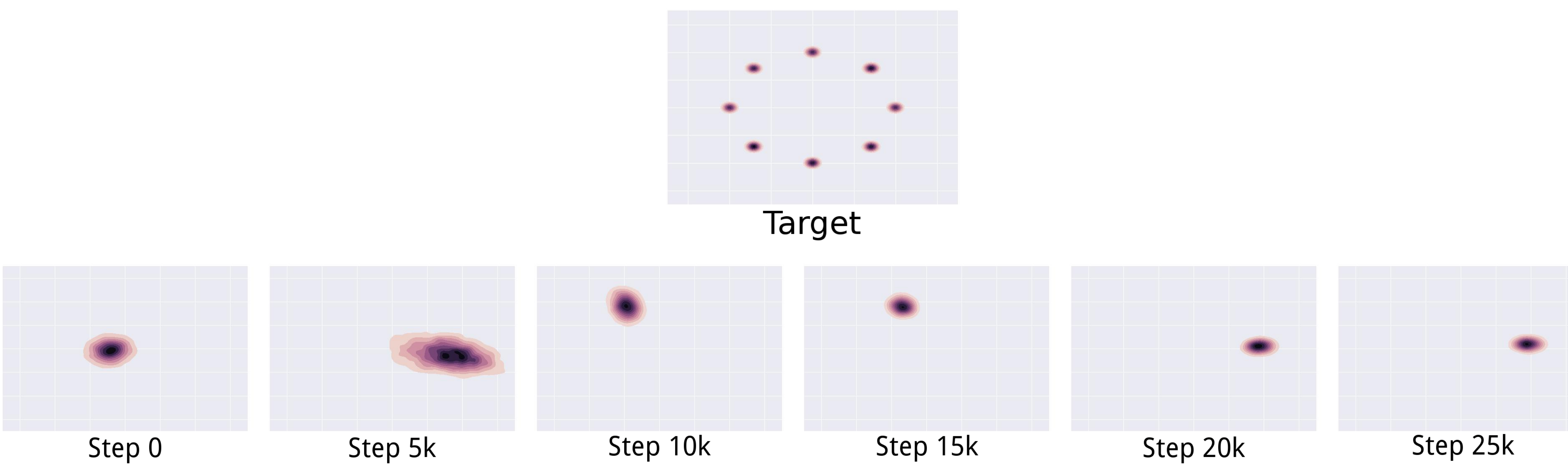
Non-convergence in GANs

- Exploiting convexity in function space, GAN training is theoretically guaranteed to converge if we can modify the density functions directly, but:
  - Instead, we modify *G* (sample generation function) and *D* (density ratio), not densities
  - We represent *G* and *D* as highly non-convex parametric functions
- “Oscillation”: can train for a very long time, generating very many different categories of samples, without clearly generating better samples
- Mode collapse: most severe form of non-convergence

Mode Collapse

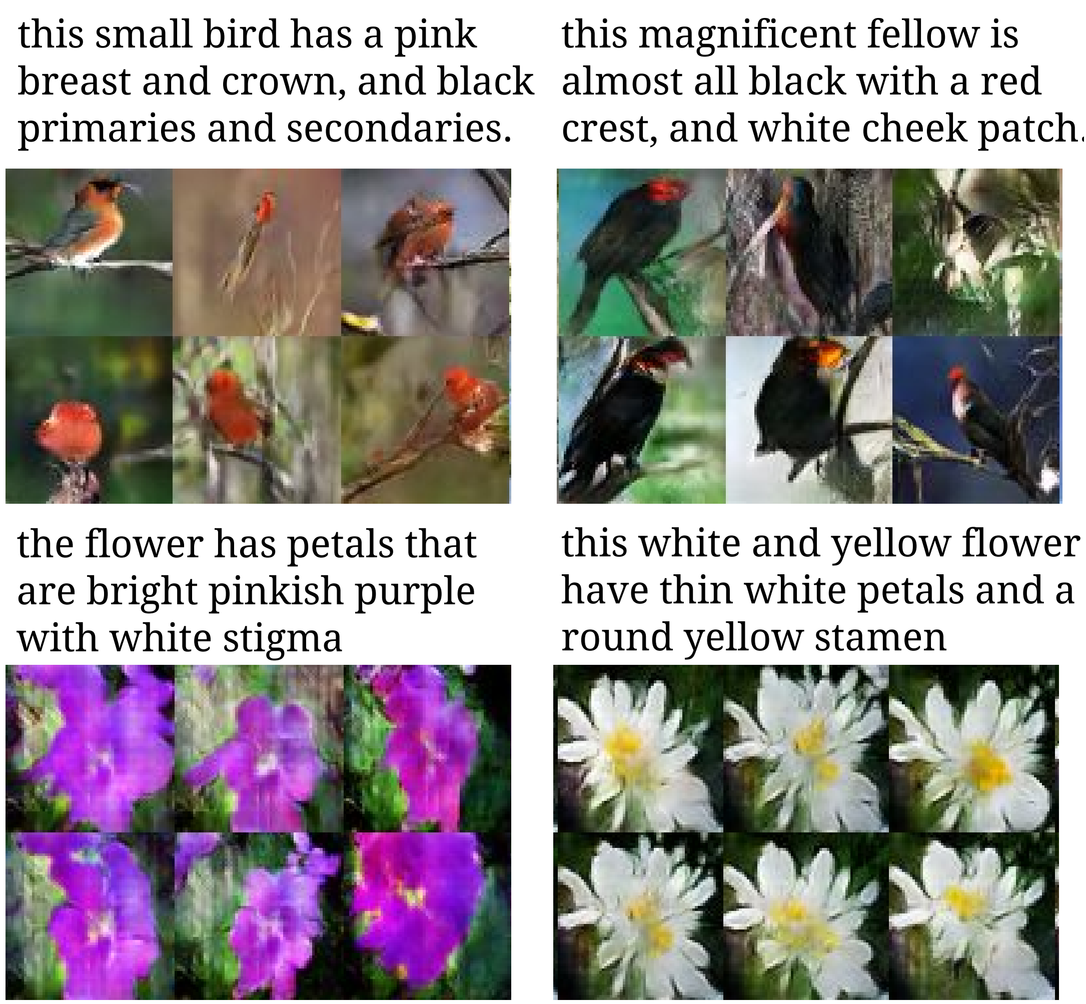
$$\min_G \max_D V(G, D) \neq \max_D \min_G V(G, D)$$

- *D* in inner loop: convergence to correct distribution
- *G* in inner loop: place all mass on most likely point



(Metz et al 2016)

Mode collapse causes low output diversity



(Reed et al 2016)

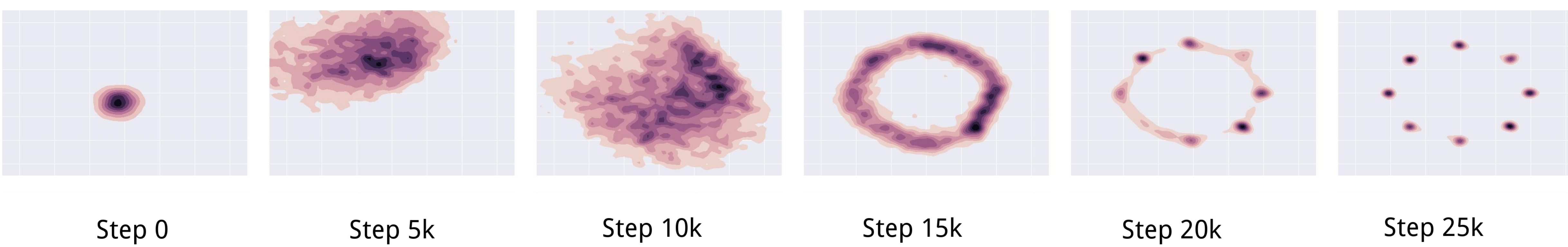
(Reed et al, submitted to ICLR 2017)

Minibatch Features

- Add minibatch features that classify each example by comparing it to other members of the minibatch (Salimans et al 2016)
- Nearest-neighbor style features detect if a minibatch contains samples that are too similar to each other

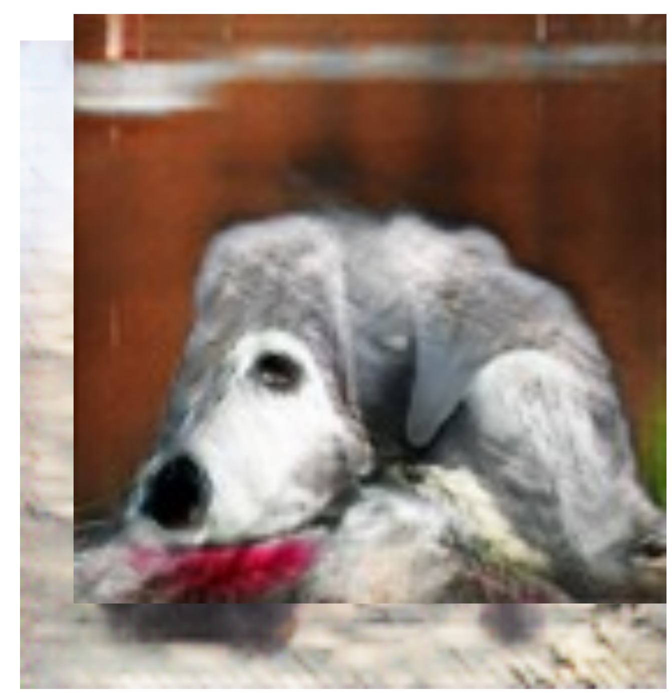
Unrolled GANs

- Backprop through *k* updates of the discriminator to prevent mode collapse:



(Metz et al 2016)

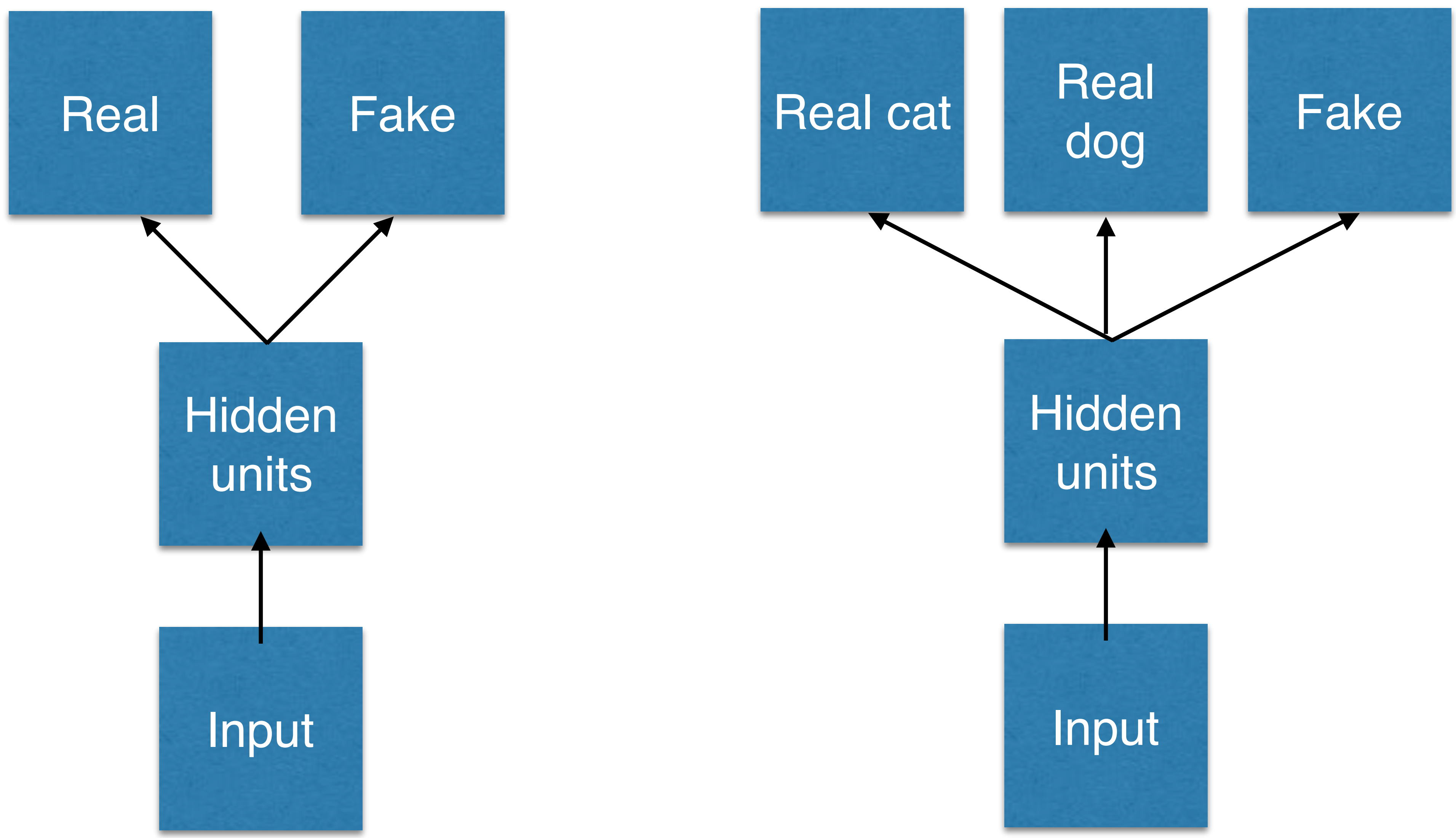
Some examples are strange!



Evaluation

- There is not any single compelling way to evaluate a generative model
    - Models with good likelihood can produce bad samples
    - Models with good samples can have bad likelihood
    - There is not a good way to quantify how good samples are
- For GANs, it is also hard to even estimate the likelihood
- See “A note on the evaluation of generative models,” Theis et al 2015, for a good overview

Supervised Discriminator



(Odena 2016, Salimans et al 2016)

Conclusion

- GANs are generative models that use supervised learning to approximate an intractable cost function
- GANs can simulate many cost functions, including the one used for maximum likelihood
- Many potential applications to explore beyond image generation!