Probabilistic (Graphical) Models

and inference

Oliver Obst · Autumn 2023

WESTERN SYDNEY
UNIVERSITY

Probabilistic (Graphical) Models and Inference

(PGM: Probabilistic Graphical Models: Principles and Techniques by Daphne Koller and Nir Friedman. MIT Press)
(PMLI: Probabilistic Machine Learning: An introduction by Kevin Murphy. MIT Press)

Week	Lecture	Required reading	Assessment
1 Monday, 4 March 2024	Introduction, Probability Theory	PGM Chapter 2, PMLI Chapter 6.1	
2 Monday, 11 March 2024	Directed and undirected networks introduction	PGM Chapter 3 & 4	Quiz 1
3 Monday, 18 March 2024	Variable elimination	PGM Chapter 9	
4 Monday, 25 March 2024	Belief propagation	PGM Chapter 10/11	Quiz 2
5 Monday, 1 April 2024	public holiday		5 April 2024: census date
6 Monday, 8 April 2024	Message passing / Graph neural networks	https://distill.pub/2021/gnn-intro/	
7 Monday, 15 April 2024	Sampling	PGM Chapter 12	Quiz 3
8 Monday, 22 April 2024	Mid-term break		
9 Monday, 29 April 2024	Variational inference	https://leimao.github.io/article/Introduction-to-Variational-Inference/	Intra-session exam
10 Monday, 6 May 2024	Autoregressive models	https://sites.google.com/view/berkeley-cs294-158-sp20/home	Quiz 4
11 Monday, 13 May 2024	Variational Auto-Encoders	https://lilianweng.github.io/posts/2018-08-12-vae/	
12 Monday, 20 May 2024	GANs	https://arxiv.org/abs/1701.00160	Quiz 5
13 Monday, 27 May 2024	Energy-based models	https://arxiv.org/abs/2101.03288	
14 Monday, 3 June 2024	Evaluating generative models		Quiz 6
Monday, 17 June 2024			Project due

Generative Adversarial Networks (GANs)

Material in today's lecture based on:

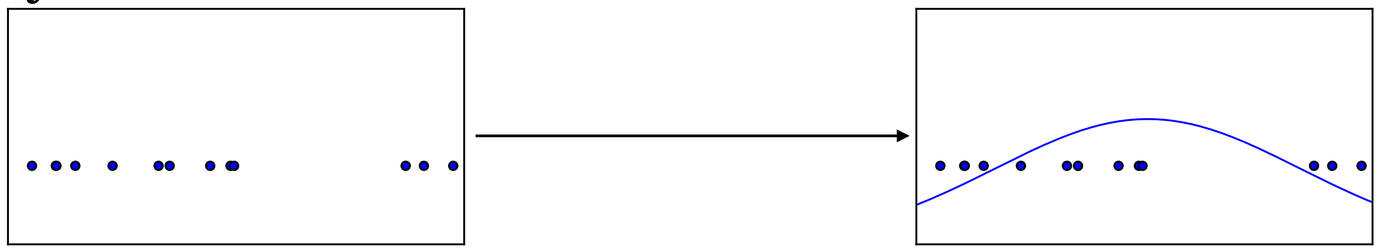
- Ian Goodfellow's tutorial on Generative Adversarial Networks

  https://arxiv.org/abs/1701.00160

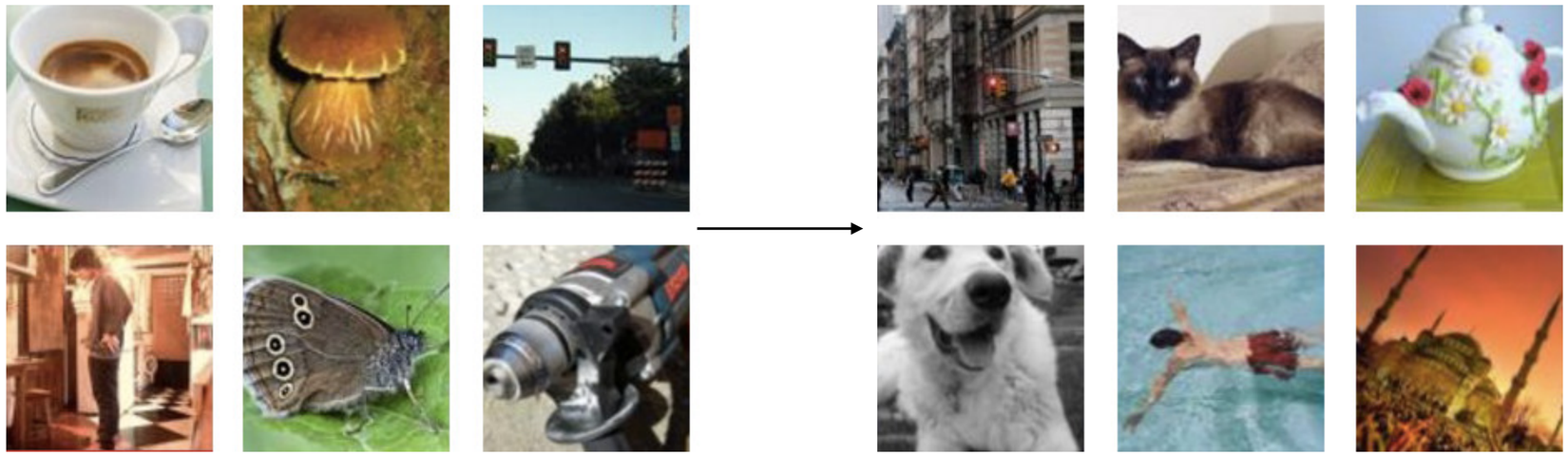
- and the slides to the tutorial, https://www.iangoodfellow.com/slides/

Generative Modeling

- Density estimation



- Sample generation



Training examples

Model samples

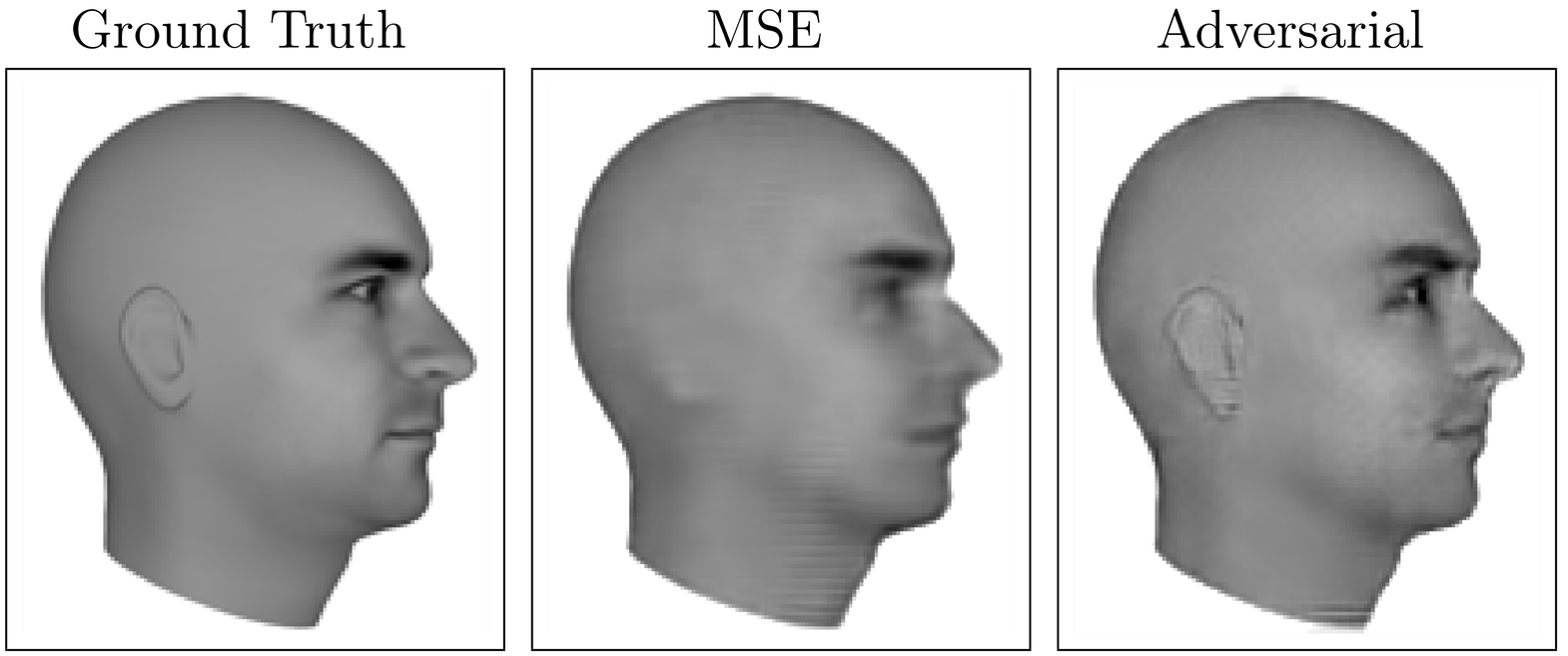
Roadmap

- Why study generative modeling?
- How do generative models work? How do GANs compare to others?
- How do GANs work?
- Tips and tricks
- Research frontiers
- Combining GANs with other methods

Why study generative models?

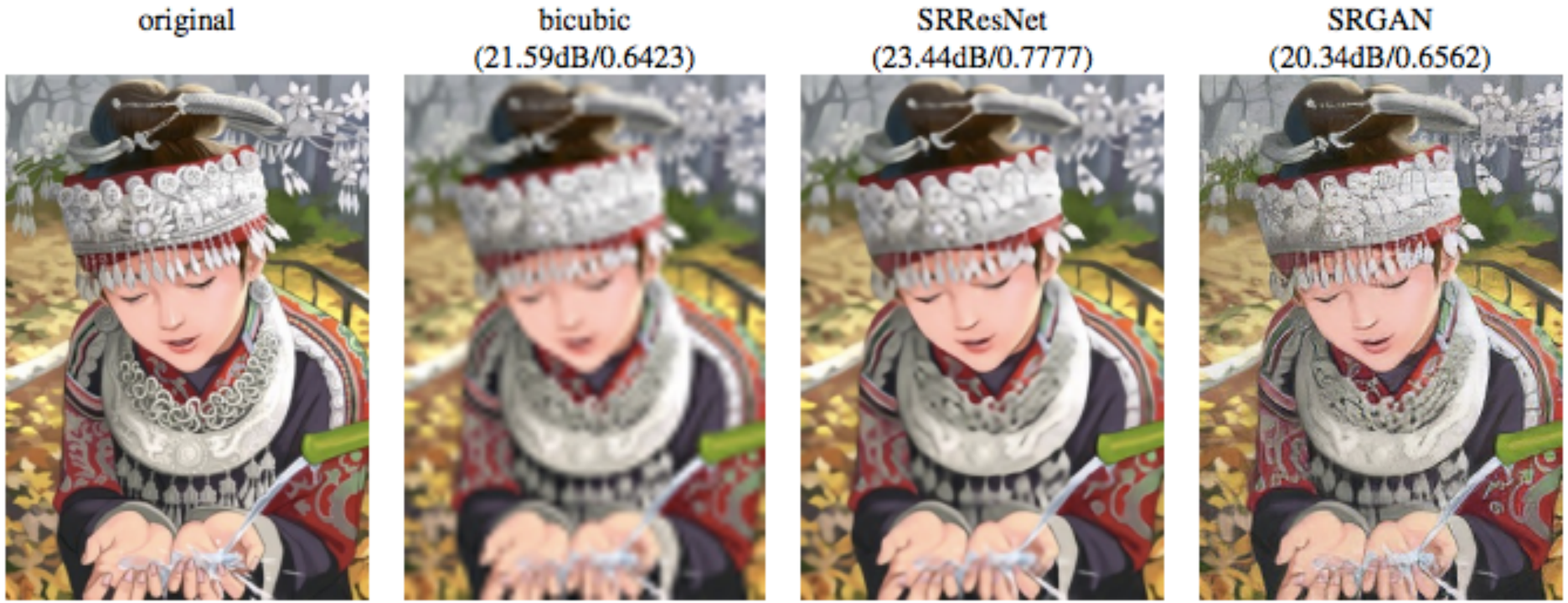
- Excellent test of our ability to use high-dimensional, complicated probability distributions
- Simulate possible futures for planning or simulated RL
- Missing data
  - Semi-supervised learning
- Multi-modal outputs
- Realistic generation tasks

Next Video Frame Prediction



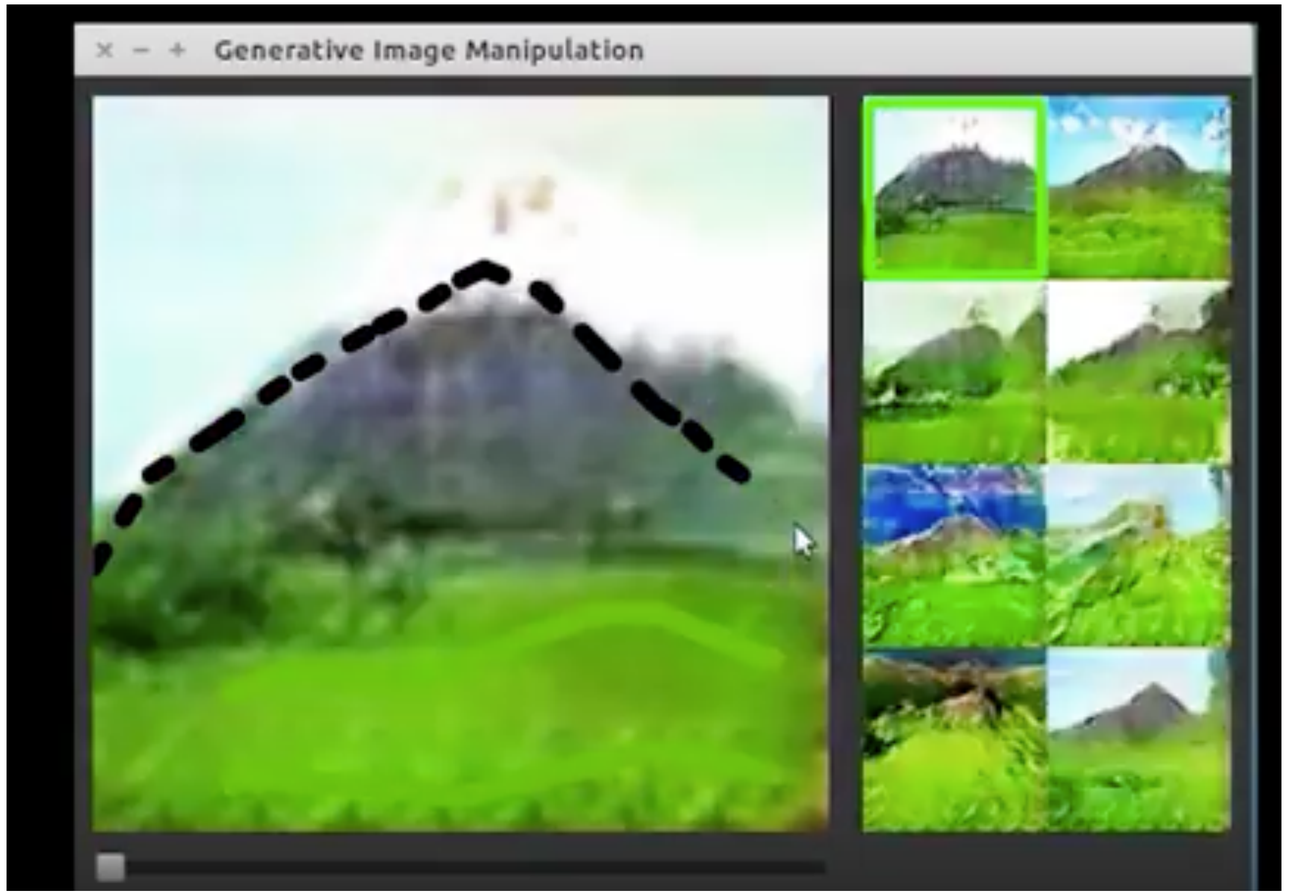
(Lotter et al 2016)

Single Image Super-Resolution



(Ledig et al 2016)

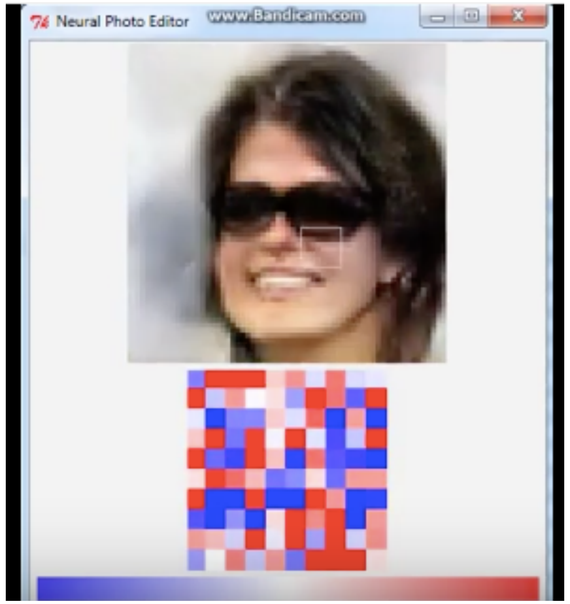
iGAN



youtube

(Zhu et al 2016)

Introspective Adversarial Networks



youtube

(Brock et al 2016)

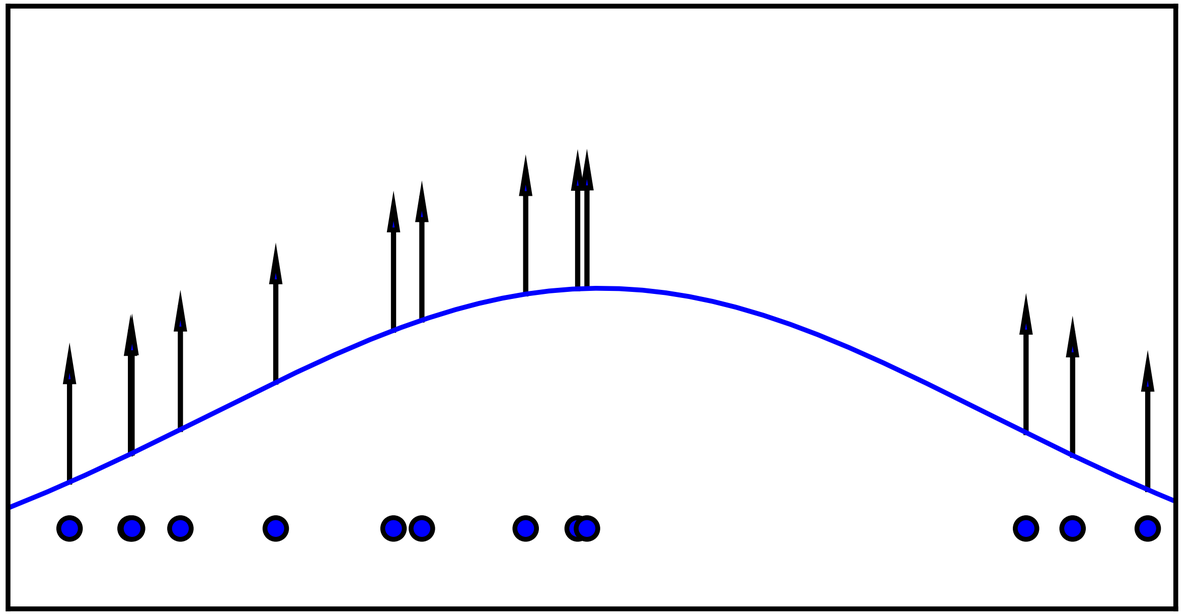
Image to Image Translation

(Isola et al 2016)

Roadmap

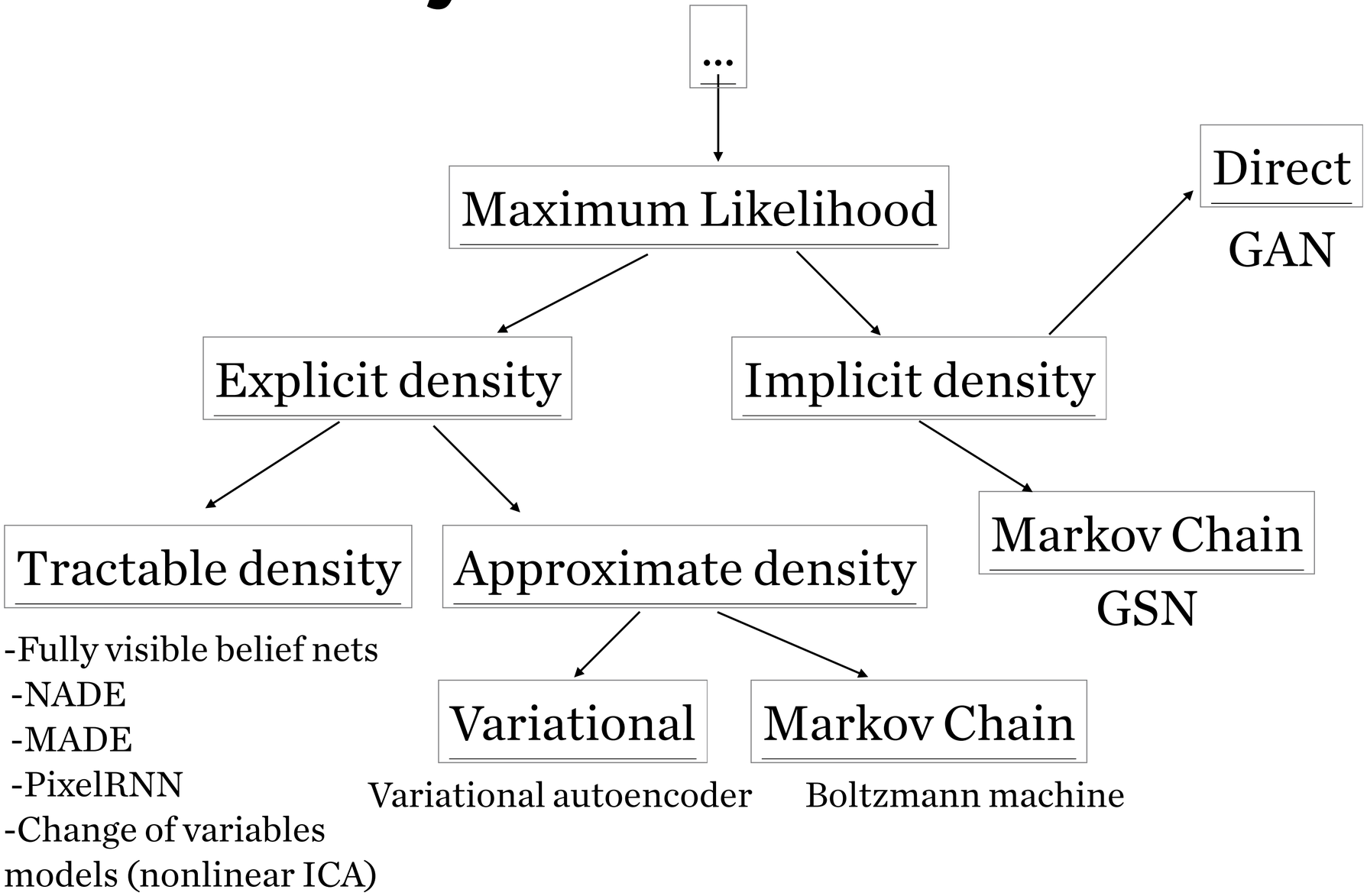
- Why study generative modeling?
- How do generative models work? How do GANs compare to others?
- How do GANs work?
- Tips and tricks
- Research frontiers
- Combining GANs with other methods

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

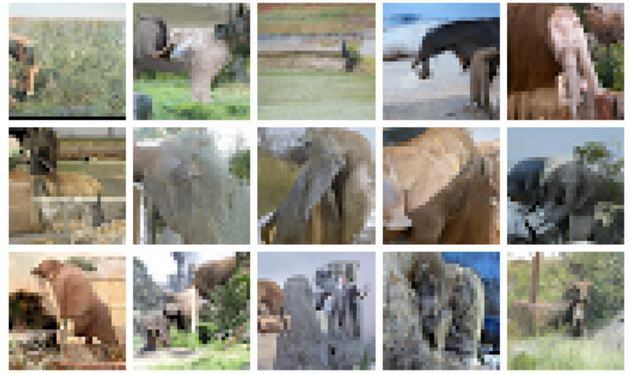


Fully Visible Belief Nets

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

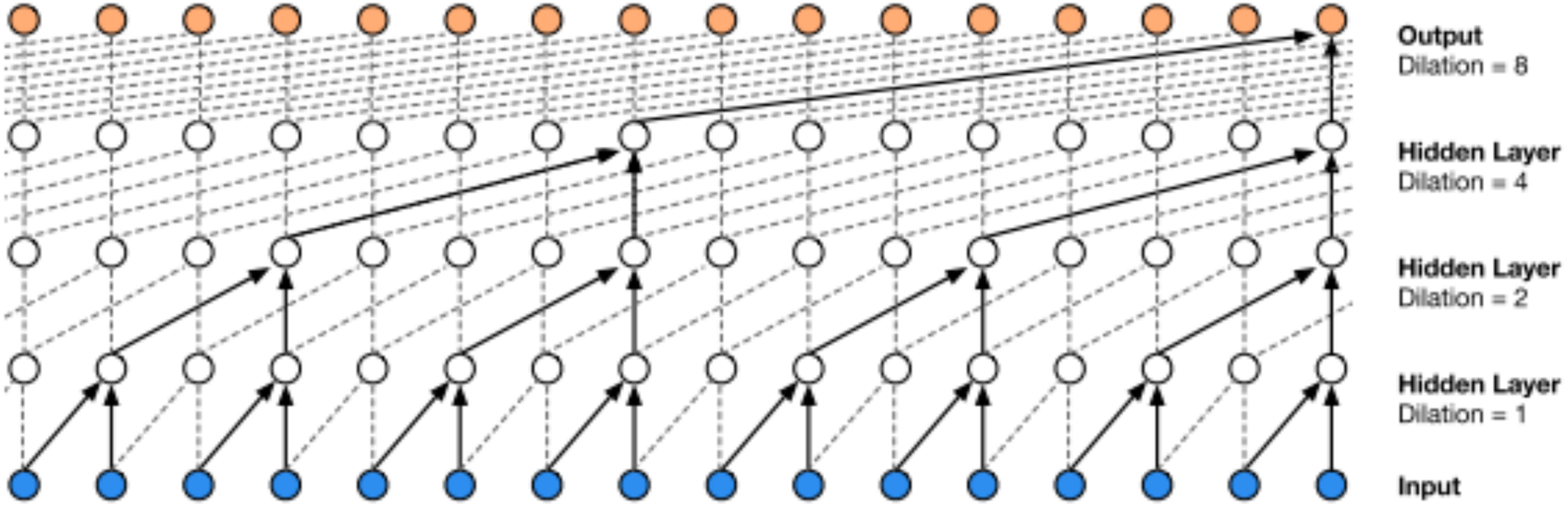
$$p_{\text{model}}(\boldsymbol{x}) = p_{\text{model}}(x_1) \prod_{i=2}^{n} p_{\text{model}}(x_i \mid x_1, \ldots, x_{i-1})$$

- 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)

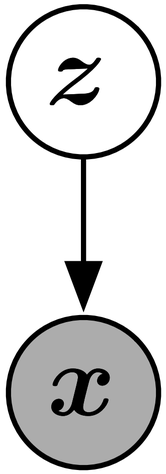
64x64 ImageNet Samples
Real NVP (Dinh et al 2016)

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

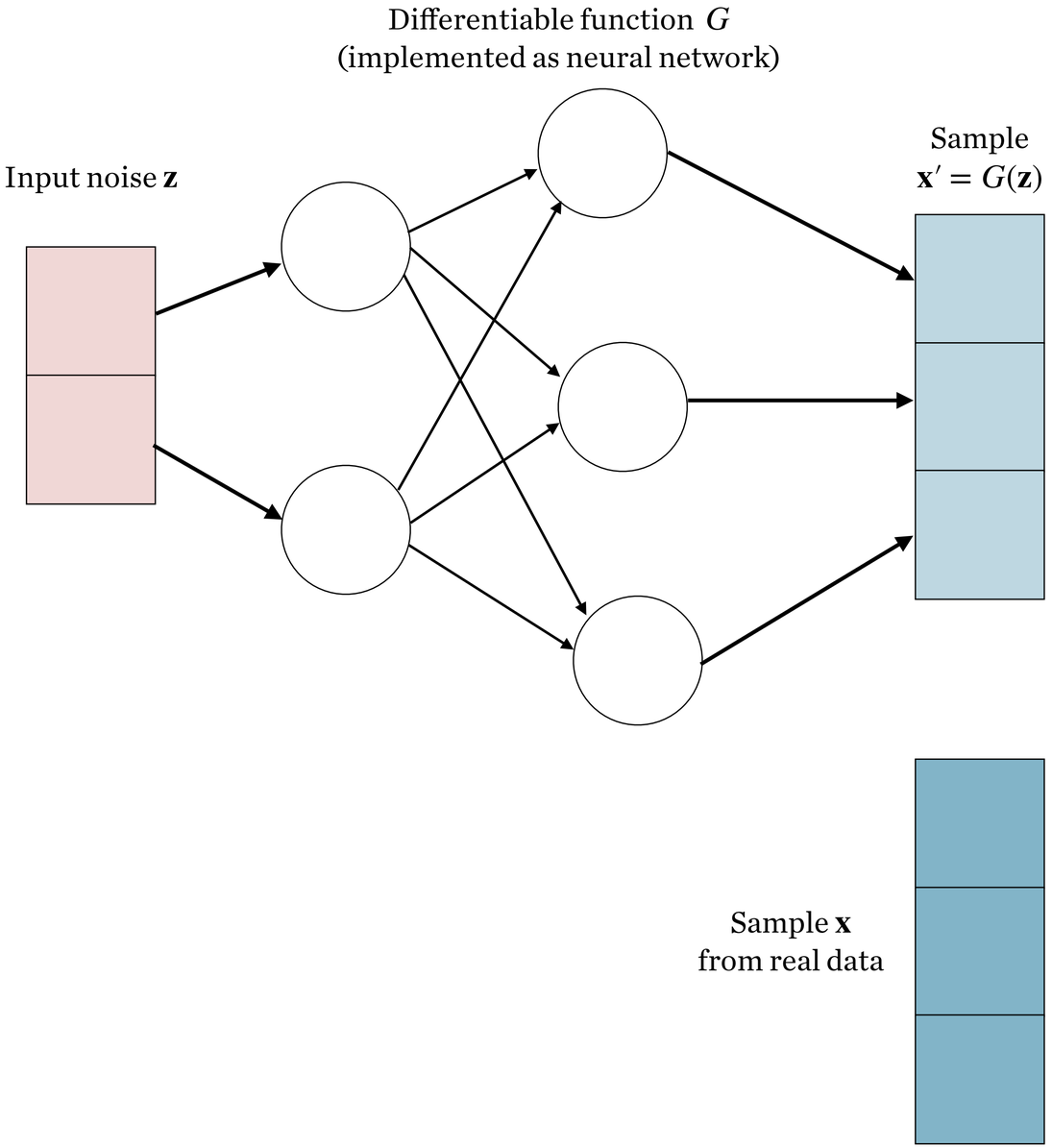
GANs

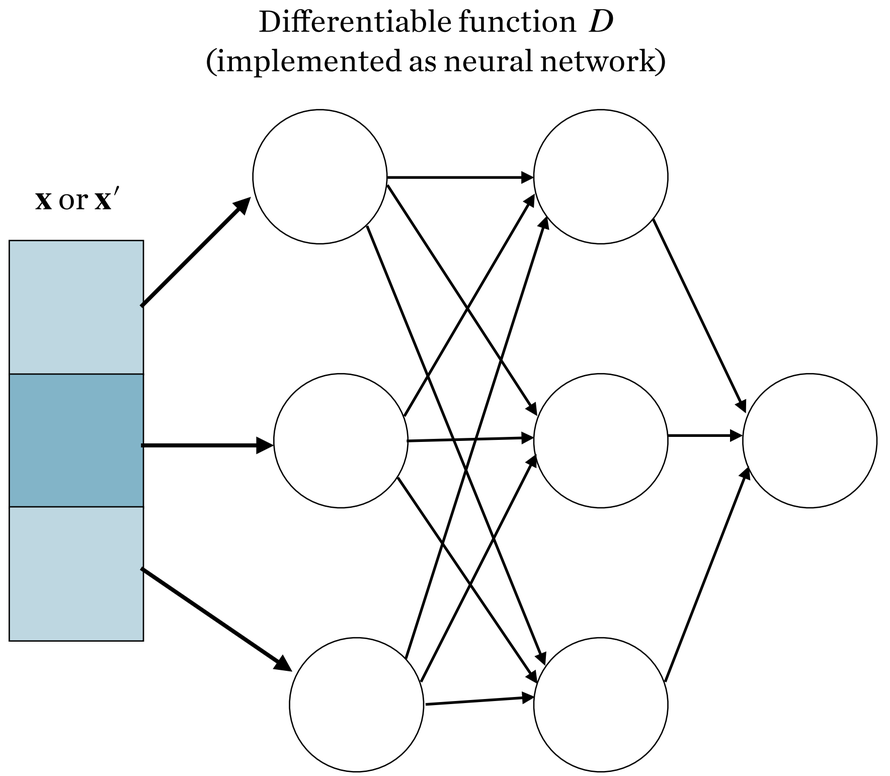
- Use a latent code
- Asymptotically consistent (unlike variational methods)
- No Markov chains needed
- Often regarded as producing the best samples
  - No good way to quantify this

Roadmap

- Why study generative modeling?
- How do generative models work? How do GANs compare to others?
- How do GANs work?
- Tips and tricks
- Research frontiers
- Combining GANs with other methods

Adversarial Networks setup





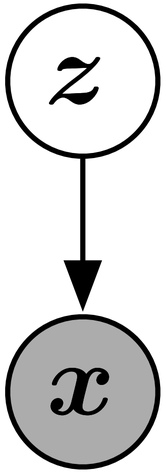
D tries $D(\mathbf{x}) \rightsquigarrow 1$
and $D(\mathbf{x}') \rightsquigarrow 0$.

G tries $G(\mathbf{x}') \rightsquigarrow 1$.

$D(\mathbf{x}') = D(G(\mathbf{z}))$

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-like algorithm of choice (Adam) 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.

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

for number of training iterations **do**

  for k steps **do**

  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
  - Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right) \right].$$

  end for

  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

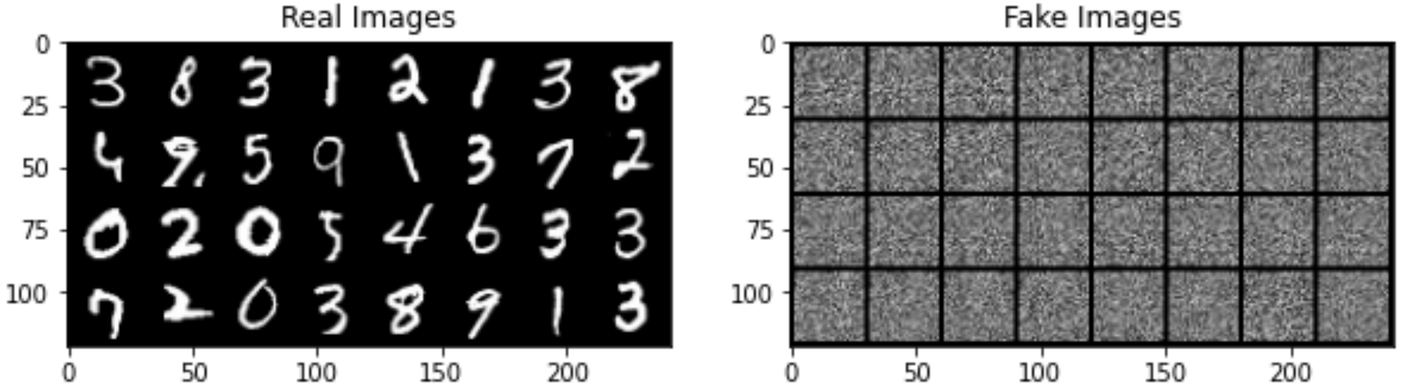
end for

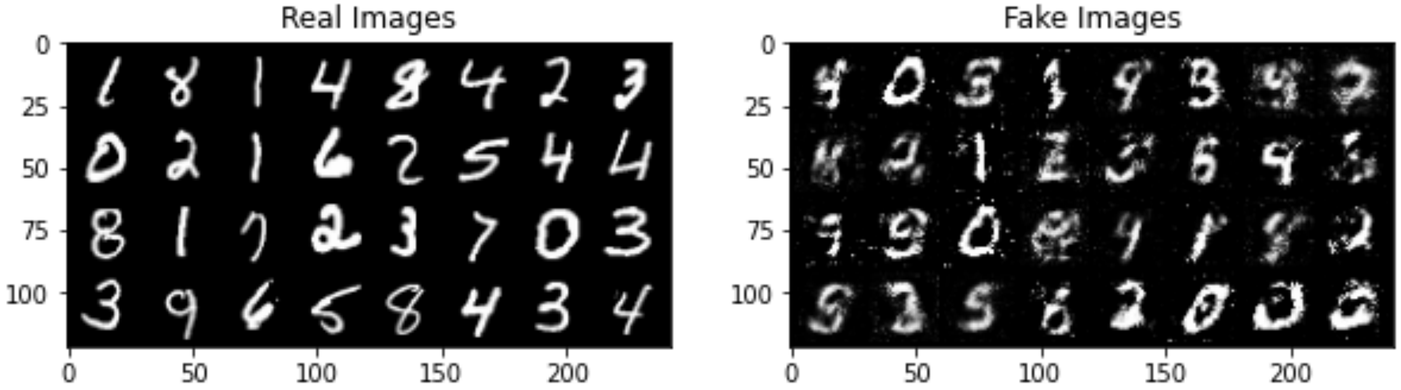
The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

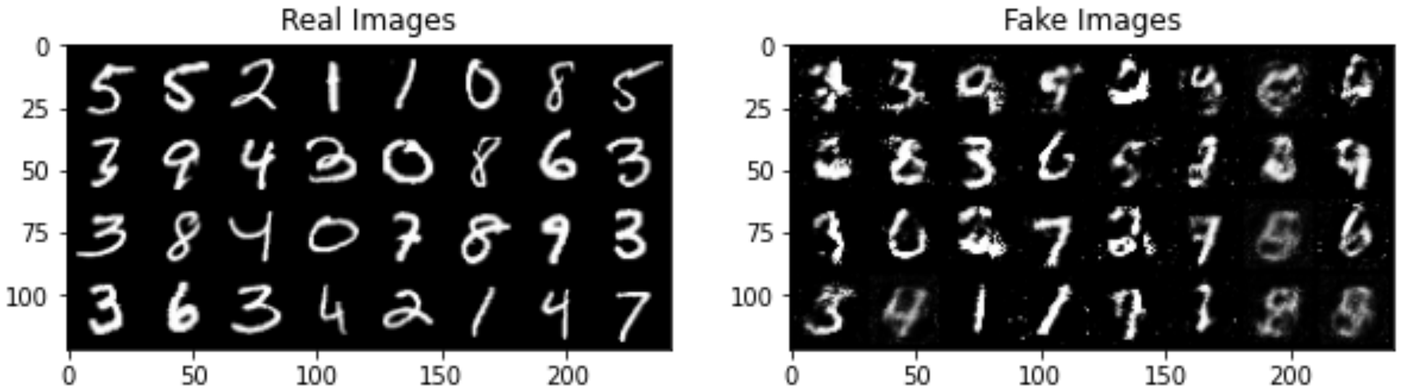
```
# Checking if we have a GPU
device = 'cuda' if torch.cuda.is_available() else 'cpu'

class Generator(nn.Module):
    def __init__(self, latent_dim, img_dim):
        super(Generator, self).__init__()
        self.gen = nn.Sequential(
            nn.Linear(latent_dim, 256),
            nn.ReLU(),
            nn.Linear(256, img_dim),
            nn.Tanh(),  # Because we normalise images to be between -1 and 1
        )

    def forward(self, x):
        return self.gen(x)


class Discriminator(nn.Module):
    def __init__(self, img_dim):
        super(Discriminator, self).__init__()
        self.disc = nn.Sequential(
            nn.Linear(img_dim, 256),
            nn.ReLU(),
            nn.Linear(256, 1),
            nn.Sigmoid(),
        )

    def forward(self, x):
        return self.disc(x)
```

```
# Initialise generator and discriminator
generator = Generator(latent_dim, img_dim).to(device)
discriminator = Discriminator(img_dim).to(device)

# optimisers
opt_gen = torch.optim.Adam(generator.parameters(), lr=lr)
opt_disc = torch.optim.Adam(discriminator.parameters(), lr=lr)

# Loss function
criterion = nn.BCELoss()

# Load Data
dataset = MNIST(root='.', transform=transforms.Compose([
    transforms.ToTensor(),
    transforms.normalise((0.5,), (0.5,))  # normalise to range [-1,1]
]), download=True)
loader = DataLoader(dataset, batch_size=batch_size, shuffle=True)
```

```
for epoch in range(epochs):
    for batch_idx, (real, _) in enumerate(loader):
        real = real.view(-1, 784).to(device)
        batch_size = real.shape[0]

        ### Train Discriminator: max log(D(x)) + log(1 - D(G(z)))
        noise = torch.randn(batch_size, latent_dim).to(device)
        fake = generator(noise)

        disc_real = discriminator(real).view(-1)
        lossD_real = criterion(disc_real, torch.ones_like(disc_real))
        disc_fake = discriminator(fake.detach()).view(-1)
        lossD_fake = criterion(disc_fake, torch.zeros_like(disc_fake))
        lossD = (lossD_real + lossD_fake) / 2
        discriminator.zero_grad()
        lossD.backward()
        opt_disc.step()

        ### Train Generator: min log(1 - D(G(z))) <-> max log(D(G(z))
        output = discriminator(fake).view(-1)
        lossG = criterion(output, torch.ones_like(output))
        generator.zero_grad()
        lossG.backward()
        opt_gen.step()
```


Real Images
Fake Images


Real Images
Fake Images


Real Images
Fake Images

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
-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 $D(x)$ 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:

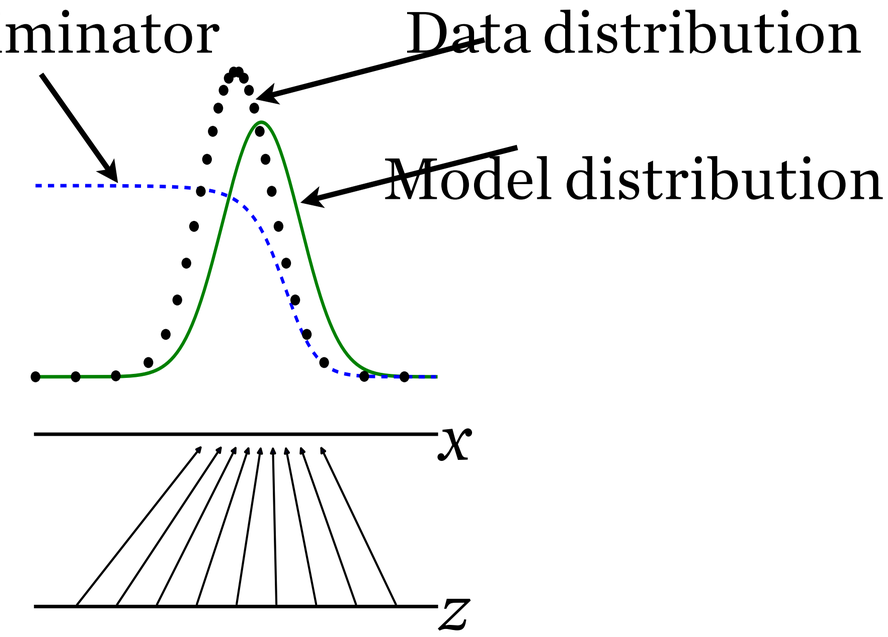
$$\frac{\delta}{\delta D(\boldsymbol{x})} J^{(D)} = 0$$

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

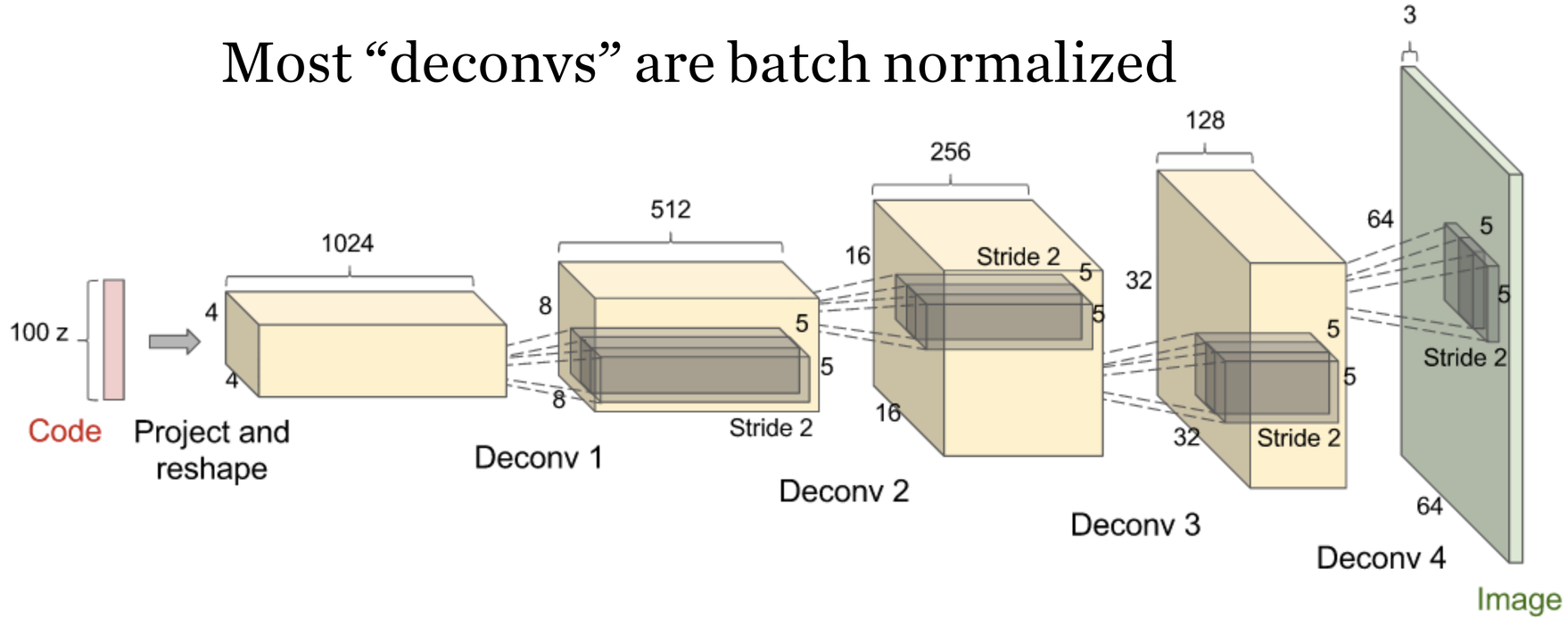
$$J^{(D)} = -\frac{1}{2}\mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}} \log D(\boldsymbol{x}) - \frac{1}{2}\mathbb{E}_{\boldsymbol{z}} \log\left(1 - D\left(G(\boldsymbol{z})\right)\right)$$

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

-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

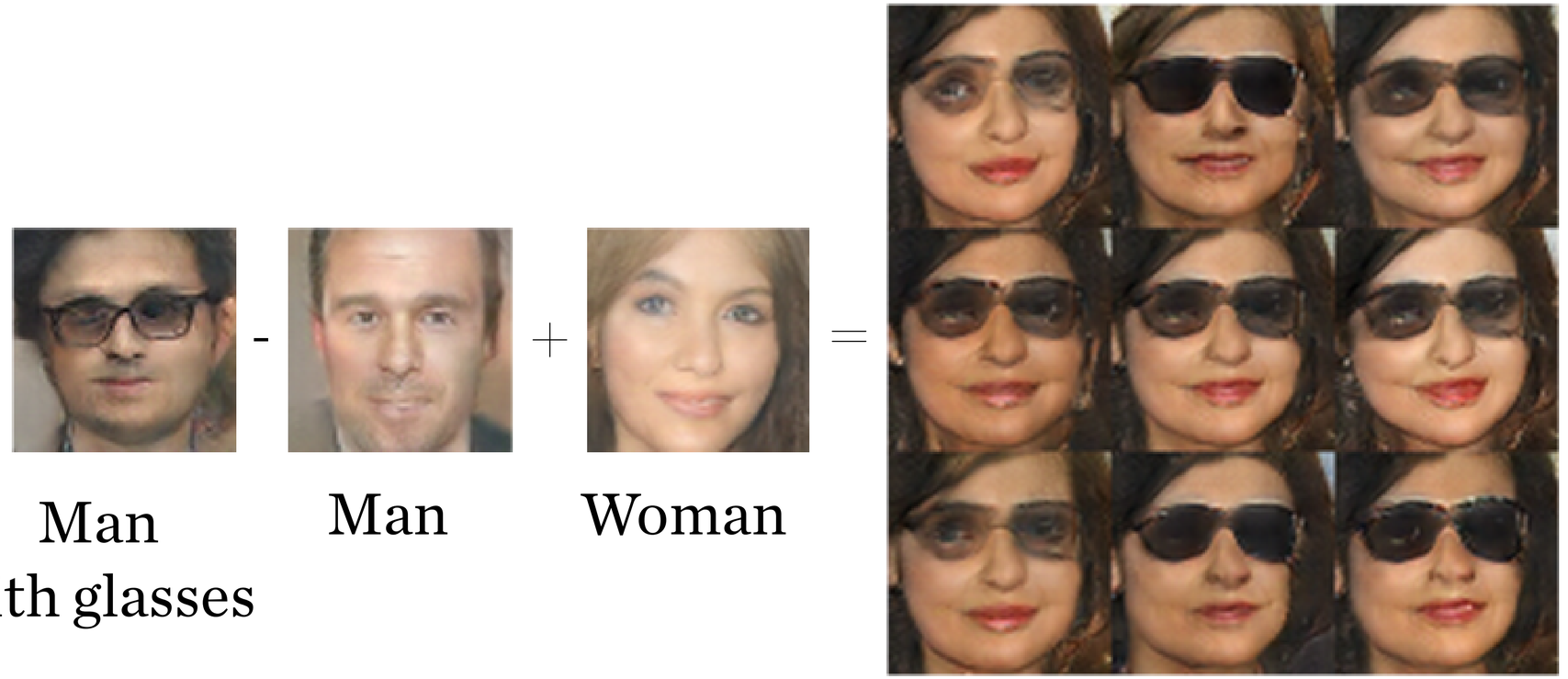


(Radford et al 2015)

DCGANs for LSUN Bedrooms

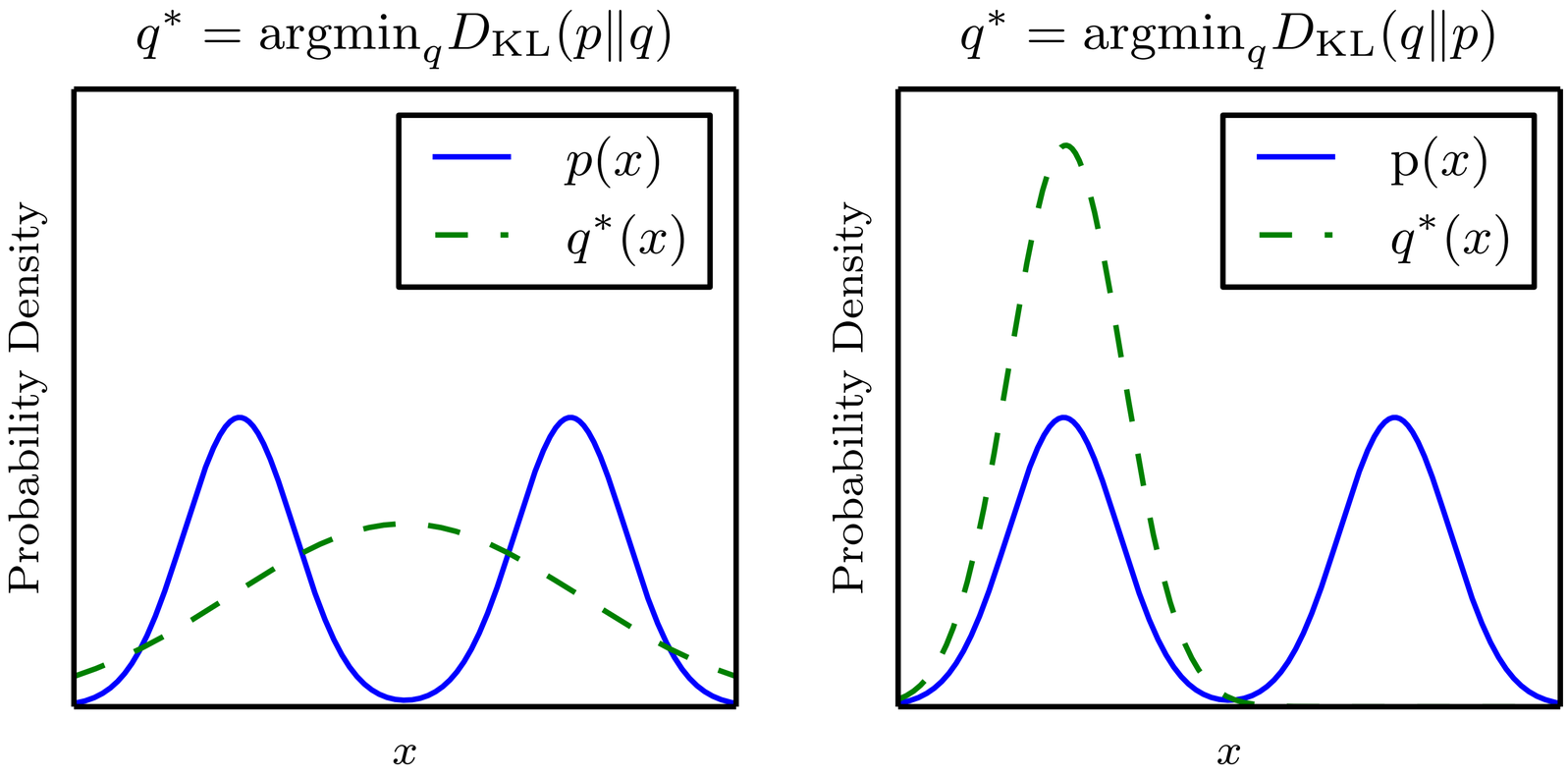
(Radford et al 2015)

Vector Space Arithmetic



(Radford et al, 2015)

Is the divergence important?



Maximum likelihood

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)

Reducing GANs to RL

- Generator makes a sample
- Discriminator evaluates a sample
- Generator's cost (negative reward) is a function of $D(G(z))$
- Note that generator's cost does not include the data, x
- Generator's cost is always monotonically decreasing in $D(G(z))$
- Different divergences change the location of the cost's fastest decrease

Roadmap

- Why study generative modeling?
- How do generative models work? How do GANs compare to others?
- How do GANs work?
- Tips and tricks
- Research frontiers
- Combining GANs with other methods

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 $X=\{x^{(1)}, x^{(2)}, .., x^{(m)}\}$
- 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* $R=\{r^{(1)}, r^{(2)}, .., r^{(m)}\}$
- Given new inputs $X=\{x^{(1)}, x^{(2)}, .., x^{(m)}\}$
- 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 $x^{(i)}$ 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* $R=\{r^{(1)}, r^{(2)}, .., r^{(m)}\}$
- Given new inputs $X=\{x^{(1)}, x^{(2)}, .., x^{(m)}\}$
- For each $x^{(i)}$ in X:
  - Construct a *virtual batch* V containing both $x^{(i)}$ and all of R
  - Compute mean and standard deviation of features of V
  - Normalize the features of $x^{(i)}$ 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

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
- Finding Nash equilibria in high-dimensional, continuous, non-convex games is an important open research problem
- GANs are a key ingredient of PPGNs, which are able to generate compelling high resolution samples from diverse image classes