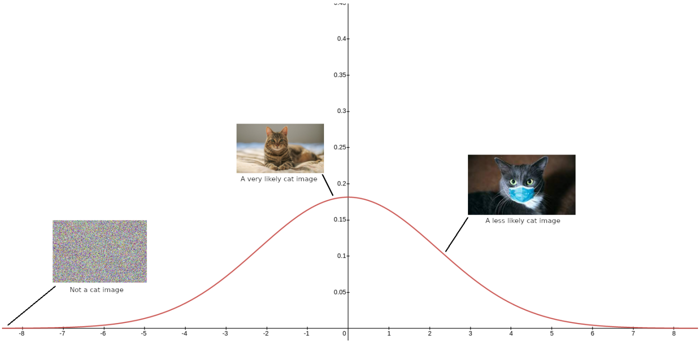
Overview of GANs

Beau Horenberger

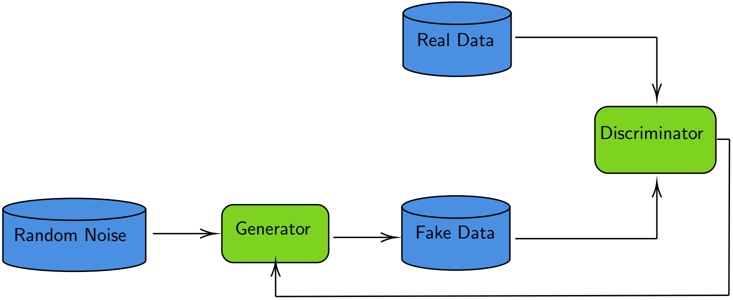
April 30, 2021

Motivating GANs

- Have samples from distribution
- Want to imitate distribution



GAN Structure



Defining a GAN

$$W = \{\mathbb{R}^n \text{ ; Random noise}\},\ X = \{\mathbb{R}^m \text{ ; Real and fake samples}\},$$
$$Y = [0, 1]$$

The discriminator h_D and generator h_G are predictors,

$$h_G : W \rightarrow X$$
$$h_D : X \rightarrow Y$$

Define random variables $\mathbf{w}$ over W and $\mathbf{x}$ over X, each with distributions $p_{\mathbf{w}}(w)$ and $p_{\mathbf{x}}(x)$, respectively. Define $p_G(x)$ as the probability distribution for $h_G(\mathbf{w})$.
h_D and h_D are multilayer perceptrons using sigmoid cross-entropy loss.

The Minimax Problem

$$\min_{h_G} \max_{h_D} V(h_D, h_G) = \min_{h_G} \max_{h_D} \left[\mathop{\mathbb{E}}_{x \sim \mathbf{x}} [\log h_D(x)] + \mathop{\mathbb{E}}_{w \sim \mathbf{w}} [\log(1 - h_D(h_G(w)))] \right]$$

GANs Minimize Jensen-Shannon Divergence

$$D_{JS}(P \parallel Q) = \frac{1}{2}D_{KL}(P \parallel \frac{1}{2}(P+Q)) + \frac{1}{2}D_{KL}(Q \parallel \frac{1}{2}(P+Q))$$

- JS Divergence is symmetric, balances possible problems from KL Divergence
- We can rephrase the minimax problem in terms of JS Divergence

$$\min_{h_G} \max_{h_D} V(h_D, h_G) = \min_{h_G} [D_{KL}(p_{\mathbf{x}} \parallel p_{\mathbf{x}} + p_G) + D_{KL}(p_G \parallel p_{\mathbf{x}} + p_G)]$$

GANs Work... In Theory...

Theorem

$\min \max V(h_D, h_G)$ *reaches a global minimum with respect to* h_G *if and only if* $p_G = p_{\mathbf{x}}$.

- Follows from the JS Divergence definition of $V(h_D, h_G)$
- Goodfellow also showed that a simple gradient descent algorithm will converge

GANs Have a Problem

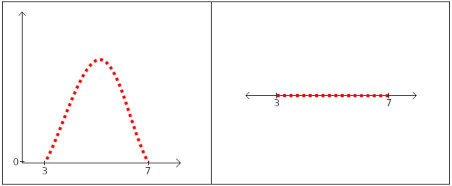
The minimax problem

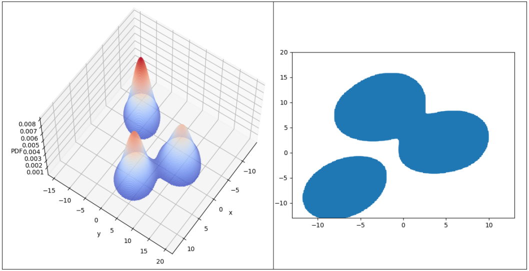
$$\min_{h_G} \max_{h_D}$$

implies optimizing h_D for each given h_G. In practice, this is bad! We usually only update h_D slightly for each udpate of h_G. Why does this occur?

The Concept of Supports

Define the *support* of a function $f : A \to B$ as the set $\{a \in A : f(a) > 0\}$.





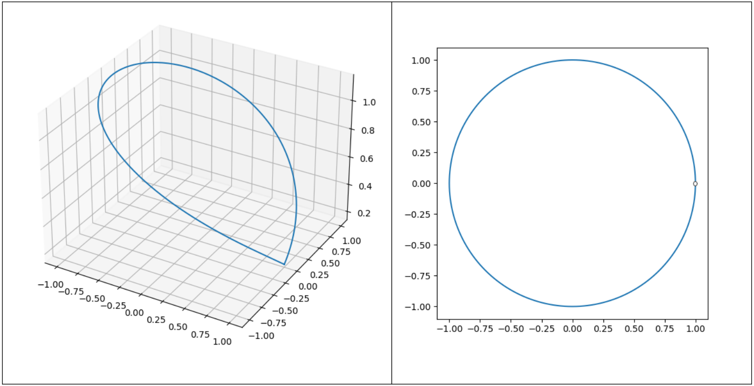
The Support of p_G Has Low Dimension

Theorem

Let $h_G : W \to X$ be a function composed by affine transformations and pointwise nonlinearities, which can either be rectifiers, leaky rectifiers, or smooth strictly increasing functions (such as the sigmoid, tanh, softplus, etc). Then $h_G(W)$ is contained in a countable union of manifolds of dimension at most $\dim W$. Therefore, if the dimension of W is less than the one of X, then $h_G(W)$ will be a set of measure 0 in X.

Examples of Low Dimensional Supports

$$p_{Gex}(r,\theta) = \begin{cases} \frac{1}{2\pi} + \sin\frac{1}{2}\theta, & \text{if } r = 1 \\ 0, & \text{otherwise} \end{cases}$$



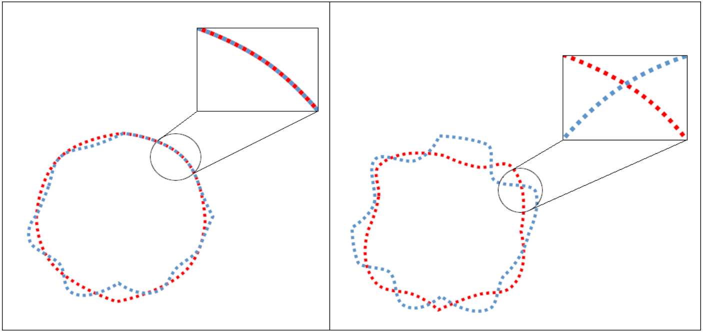
The Support of p_x Also Has Low Dimension

- The *manifold hypothesis* is the assumption that real-world data is typically a lower-dimensional manifold embedded in higher dimensional space

- Facial images are approximately 1/2 the dimension of the space they are embedded in

The Intersection of p_G and $p_{\mathbf{x}}$ is Probably 0

When two low-dimensional manifolds are perturbed, the remaining intersection probably has measure 0.



The Consequence: Perfect Discriminators Usually Exist

- When the intersection of p_G and $p_{\mathbf{x}}$ has measure 0, the discriminator will be effectively perfect.
- It can also be shown that, as the discriminator becomes perfect, then the gradients either vanish or diverge!
- This is why training empirically suffers when the discriminator is trained too much

Overcoming the Problems of GANs

- Many modifications to GANs are being explored
- Using different loss functions or f-divergences
- Combining with other methods, such as convolution (DCGAN)
- New distance measures, i.e. Wasserstein GANs

Sample GAN Code

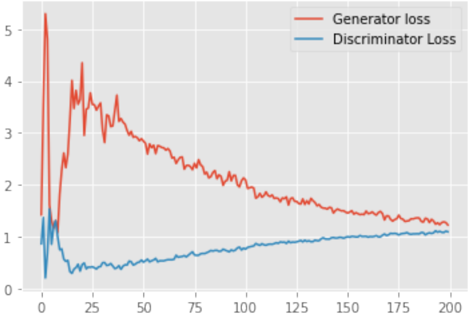
```python
class Discriminator(nn.Module):
    def __init__(self):
        super(Discriminator, self).__init__()
        self.n_input = 784
        self.main = nn.Sequential(
            nn.Linear(self.n_input, 1024),
            nn.LeakyReLU(0.2),
            nn.Dropout(0.3),

            nn.Linear(1024, 512),
            nn.LeakyReLU(0.2),
            nn.Dropout(0.3),

            nn.Linear(512, 256),
            nn.LeakyReLU(0.2),
            nn.Dropout(0.3),

            nn.Linear(256, 1),
            nn.Sigmoid(),
        )

    def forward(self, x):
        x = x.view(-1, 784)
        return self.main(x)
```

(a) Discriminator Code

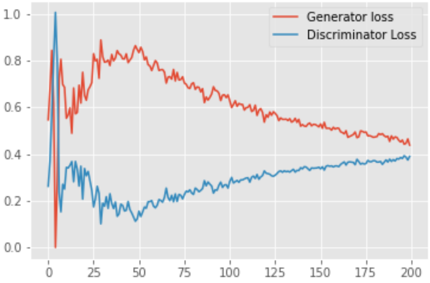
```python
class Generator(nn.Module):
    def __init__(self, nz):
        super(Generator, self).__init__()
        self.nz = nz
        self.main = nn.Sequential(
            nn.Linear(self.nz, 256),
            nn.LeakyReLU(0.2),

            nn.Linear(256, 512),
            nn.LeakyReLU(0.2),

            nn.Linear(512, 1024),
            nn.LeakyReLU(0.2),

            nn.Linear(1024, 784),
            nn.Tanh(),
        )

    def forward(self, x):
        return self.main(x).view(-1, 1, 28, 28)
```

(b) Generator Code

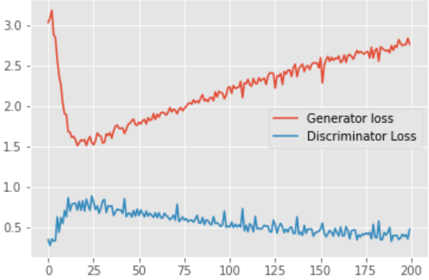
Outputs from GAN on MNIST



Outputs from LSGAN on MNIST



Outputs from DCGAN on MNIST



Generated Cats

(a) Real Cat Images

(b) Generated Cat Images

Conclusion

- GANs are a great innovation in generative methods
- GANs have fundamental problems with vanishing/exploding gradients
- Innovations upon GANs have theoretical room for improvement
- Thank you!