Topics for the week

- Transformers
- GNNs
- VAEs
- GANs
- Connecting the dots
The problem

• From a large collection of images of faces, can a network learn to *generate* new portrait
  – Generate samples from the distribution of “face” images
    • How do we even characterize this distribution?

Try visiting: https://thispersondoesnotexist.com
# Discriminative vs Generative Models

Given a distribution of inputs $X$ and labels $Y$.

<table>
<thead>
<tr>
<th>Discriminative models</th>
<th>Generative models</th>
</tr>
</thead>
<tbody>
<tr>
<td>Discriminative models learn conditional distribution $P(Y</td>
<td>X)$</td>
</tr>
<tr>
<td>Learns decision boundary between classes.</td>
<td>Learns actual probability distribution of data.</td>
</tr>
<tr>
<td>Limited scope. Can only be used for classification tasks.</td>
<td>Can do both generative and discriminative tasks.</td>
</tr>
<tr>
<td>E.g. Logistic regression, SVM etc.</td>
<td>E.g. Naïve Bayes, Gaussian Mixture Model etc.</td>
</tr>
<tr>
<td></td>
<td>Harder problem, requires a deeper understanding of the distribution than discriminative models.</td>
</tr>
</tbody>
</table>
What we have seen: VAE

- The decoder of a VAE is a generator!
- \( \hat{X} \rightarrow \) What input image \( X \) would have been encoded into the \( Z \) that I saw?
- Trained by maximizing the **likelihood of the data**
  - Likelihood maximization does not actually relate to whether the output *actually looks* like a face
- Can we make the training criteria a little more direct?
The problem

What’s the easiest way to check if the produced output looks like a face or not?
What’s the easiest way to check if the produced output looks like a face or not?

• You could just eyeball 👀 the results and use your understanding of what faces look like to evaluate what is generated.

• Unfortunately, you are not a differentiable function 😞

• But what if we could, find a differentiable proxy for you and your non-differentiable-ness?
What are GANs

**Generative Models** which generate data similar to the training data. E.g. Variational Autoencoders (VAE)

**Adversarial Training**
GANS are made up of two competing networks (adversaries) that are trying beat each other. A “game” is being played between the two.
What are GANs?

$Z \sim P(Z)$ → Generator $G(Z) → \hat{X}$ → Discriminator $D(X) → \text{Real/Fake?}$

Real data $X$ → Generated data $\hat{X}$
The GAN formulation

- For real data $X$, the desired output of the discriminator is $D(X) = 1$
  - The log probability that the instance is real, as computed by the discriminator is $\log D(X)$

- For synthetic data $\hat{X}$, the desired output of the discriminator is $D(\hat{X}) = 0$
  - The log probability that the instance is synthetic, as computed by the discriminator, is $\log(1 - D(\hat{X}))$
    - $= \log(1 - D(G(Z)))$
The original GAN formulation is the following min-max optimization

\[
\min_G \max_D \mathbb{E}_{x \sim P_x} \log D(x) + \mathbb{E}_{x \sim P_G} \log(1 - D(x))
\]

- **Objective of** $D$: Optimize for $D(x) = 1$ and $D(G(z)) = 0$
- **Objective of** $G$: Optimize for $D(G(z)) = 1$
The iterated learning

- Discriminator learns perfect boundary
- Generator moves its distribution past the boundary “into” the real distribution
- Discriminator relearns new “perfect” boundary
- Generator shifts distribution past new boundary
- ...
- In the limit Generator’s distribution sits perfectly on “real” distribution and the perfect discriminator is still random
Analysis of optimal behavior:
The optimal discriminator

- The **optimal discriminator** would be a Bayesian classifier

\[
D(x) = \frac{P_X(x)}{P_X(x) + P_G(x)}
\]

- Assuming uniform prior
Analysis of optimal behavior:

The optimal generator

$D(x) = \frac{P_X(x)}{P_X(x) + P_G(x)}$

- The optimal generator:
  $$\min_G 2D_{JSD}(P_X(x), P_G(x)) - \log 4$$

- The optimal generator minimizes the Jensen Shannon divergence between the distributions of the actual and synthetic data!
  - Tries to make the two distributions maximally similar
# VAEs vs GANs

<table>
<thead>
<tr>
<th>VAEs</th>
<th>GANs</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Minimizing the KL divergence between distributions of synthetic and true data</td>
<td>• Minimizing the Jenson-Shannon divergence between distributions of synthetic and true data</td>
</tr>
<tr>
<td>• Uses an encoder to predict latent distributions to optimize generator</td>
<td>• Use a discriminator to optimize generator</td>
</tr>
<tr>
<td>• More complex formulation</td>
<td>• Simpler formulation</td>
</tr>
<tr>
<td>• Simpler optimization. Trains faster and more reliably</td>
<td>• Sharper results</td>
</tr>
<tr>
<td>• Results are blurry</td>
<td>• Noisy and difficult optimization</td>
</tr>
</tbody>
</table>
Training GANs

Notation

• Data sample: $x$
• Input noise vector: $z$
• Distribution of Real Data: $P_X$
• Distribution of Generated Data: $P_G$
• Distribution of Input noise vector: $P_Z$
• Generator: $G(z; \theta_G)$
• Discriminator: $D(x; \theta_D)$
• Generator output: $G(z), \hat{x}$
• Discriminator output: $D(x), D(G(z))$
How to Train a GAN?

Step 1: Train the Discriminator using the current Generator

Step 2: Train the Generator to beat the Discriminator

Optimize: \( \min_G \max_D \mathbb{E}_{P_x} \log D(x) + \mathbb{E}_{P_z} \log(1 - D(G(z))) \)

The discriminator is not needed after convergence
Algorithm 1: Minibatch stochastic gradient descent training for GANs

\[
\text{for num\_epochs do:}
\]
\[
\text{for k\_steps do:}
\]
\[
\{z^{(1)} \ldots z^{(m)}\} \sim P_{z} \quad \text{(Sample m noise vectors)}
\]
\[
\{x^{(1)} \ldots x^{(m)}\} \sim P_{x} \quad \text{(Sample m data points)}
\]
\[
L_{D} \leftarrow \frac{1}{m} \sum_{i=1}^{m} \left[ \log D(x^{(i)}) + \log \left( 1 - D(G(z^{(i)}) \right) \right]
\]
\[
g_{\theta_{D}} \leftarrow \nabla_{\theta_{D}} L_{D}
\]
\[
\theta_{D} \leftarrow \theta_{D} + \alpha \cdot g_{\theta_{D}}
\]
\[
\text{end for}
\]
\[
\{z^{(1)} \ldots z^{(m)}\} \sim P_{z} \quad \text{(Sample m noise vectors)}
\]
\[
L_{G} \leftarrow \frac{1}{m} \sum_{i=1}^{m} \log \left( 1 - D(G(z^{(i)})) \right)
\]
\[
g_{\theta_{G}} \leftarrow \nabla_{\theta_{G}} L_{G}
\]
\[
\theta_{G} \leftarrow \theta_{G} - \alpha \cdot g_{\theta_{G}}
\]
\[
\text{end for}
\]

Goodfellow et. al. (2014), Generative Adversarial Networks
Training GANs

Training Algorithm

Algorithm 1: Minibatch stochastic gradient decent training for GANs

\[
\text{for num\_epochs do:} \\
\text{for k\_steps do:} \\
\quad \{z^{(1)} \ldots z^{(m)}\} \sim P_z \text{ (Sample)} \\
\quad \{x^{(1)} \ldots x^{(m)}\} \sim P_x \text{ (Sample m data points)} \\
\quad \mathcal{L}_D \leftarrow \frac{1}{m} \sum_{i=1}^m \left[ \log D(x^{(i)}) + \log \left( 1 - D(G(z^{(i)})) \right) \right] \\
\quad g_{\theta_D} \leftarrow \nabla_{\theta_D} \mathcal{L}_D \\
\quad \theta_D \leftarrow \theta_D + \alpha \cdot g_{\theta_D} \\
\text{end for} \\
\quad \{z^{(1)} \ldots z^{(m)}\} \sim P_z \text{ (Sample m noise vectors)} \\
\quad \mathcal{L}_G \leftarrow \frac{1}{m} \sum_{i=1}^m \log \left( 1 - D(G(z^{(i)})) \right) \\
\quad g_{\theta_G} \leftarrow \nabla_{\theta_G} \mathcal{L}_G \\
\quad \theta_G \leftarrow \theta_G - \alpha \cdot g_{\theta_G} \\
\text{end for}
\]

Hyperparameter. Goodfellow et al. used k=1

In practice, this saturates early in training. We can instead maximize log(D(G(z))) for better gradients.

\[
\min_G \max_D \mathbb{E}_{P_X} \log D(x) - \mathbb{E}_{P_z} \log(D(G(z)))
\]

\[
\min_G \max_D \mathbb{E}_{P_X} \log D(x) - \mathbb{E}_{P_G} \log(D(x))
\]

Goodfellow et. al. (2014), Generative Adversarial Networks
Training GANs
Theorems

\[ L = 2D_{JS}(P_X(x), P_G(x)) - \log 4 \]

**Theorem 1:** Global minima of the loss is achieved iff \( P_X = P_G \) where \( L = -\log 4 \)
and \( D(x) = 1/2 \)

**Theorem 2:** if \( G \) and \( D \) have enough capacity, and at each step of Algorithm 1, the discriminator is allowed to reach its optimum given \( G \), and \( P_G \) is updated so as to improve \( L \), then \( P_G \) converges to \( P_X \). [Proof in Section 4.2]
Training GANs
Distance Functions - KL

1. For $D_{KL}(P_X \ || \ P_G)$, red:
   1. High penalty (peak) where $P_X(x)$ is high but $P_G(x)$ is low.
   2. Low penalty where $P_X(x)$ is low but $P_G(x)$ is high.
      (It is fine to produce low quality images, but VERY BAD to drop image modes i.e. low diversity)

2. For $D_{KL}(P_G \ || \ P_X)$, blue:
   1. High penalty (peak) where $P_X(x)$ is low but $P_G(x)$ is high.
   2. Low penalty where $P_X(x)$ is high but $P_G(x)$ is low.
      (It is fine to produce samples that cover fewer image modes i.e. to have low diversity, but VERY BAD to have low quality)
Training GANs

Distance Functions - JS

\( D_{JS}(P_X, P_G) \) was supposed to handle both issues: quality and diversity.

But it didn’t quite happen.

We need to look at what exactly causes this difficulty in training GANs, and why JS-Divergence is not a good enough metric for our purpose.
Difficulty in Training GANs

Issues with the Discriminator

As the Discriminator gets more and more confident, the region that gives us informative gradients keeps shrinking.
Difficulty in Training GANs

Issues with the Discriminator

As the Discriminator gets more and more confident, the region that gives us informative gradients keeps shrinking.

… Till we eventually achieve a step-function which tells us nothing about how G should be changed for the desired effect on the model.
Difficulty in Training GANs

Uninformative Gradients

During early stages of training, when G is pretty-bad:

$$\min_G \max_D \mathbb{E}_{P_X} \log D(x) + \mathbb{E}_{P_z} \log(1 - D(G(z)))$$

$$\approx \mathbb{E}_{P_X} \log(1) + \mathbb{E}_{P_z} \log(1)$$

Not very informative..
Difficulty in Training GANs

Uninformative Gradients

How can we improve on this?

$$p(x)$$

$$P_X(x)$$  Better $$D(x)$$?  $$P_G(x)$$

$$p(x)$$

Better $$P_X(x)$$ and $$P_G(x)$$?
Let us Consider the Worst case scenario,

Let there be 2 distributions $P$ and $Q$ such that they are only 1 for 1 value of $x$ and 0 otherwise. Let $\theta$ be the distance between the peaks of these distributions.
So Far,

**VAE:** KL Divergence

\[ D_{KL}(p(x) \parallel q(x)) = \sum_{x \in X} p(x) \ln \frac{p(x)}{q(x)} \]

**GAN:** JSD

\[ D_{JS}(p \parallel q) = \frac{1}{2} D_{KL}(p \parallel \frac{p+q}{2}) + \frac{1}{2} D_{KL}(q \parallel \frac{p+q}{2}) \]
KL Divergence

\[ KL(P||Q) = \sum_x P(x) \frac{P(x)}{Q(x)} \]

\[ KL(P||Q) = \sum_x P(0) \frac{P(0)}{Q(0)} + P(1) \frac{P(1)}{Q(1)} + \cdots \]

\[ KL(P||Q) = 1 \log \frac{1}{1} + \cdots 0 + \cdots 0 = 0 \]

\[ KL(P||Q) = 1 \log \frac{1}{0} + 0 \log \frac{0}{1} + \cdots 0 = \infty \]
JS Divergence

\[ m = \frac{P + Q}{2} \]

\[ JSD(P||Q) = \frac{1}{2} KL(P||m) + \frac{1}{2} KL(Q||m) \]

\[ JSD(P||Q) = \frac{1}{2} \sum_x P(0) \frac{P(0)}{m(0)} + P(1) \frac{P(1)}{m(1)} + \cdots + \frac{1}{2} \sum_x Q(0) \frac{Q(0)}{m(0)} + Q(1) \frac{Q(1)}{m(1)} + \cdots \]

at \( x = 0 \), \quad m = \frac{1 + \frac{1}{2}}{2} = 1

\[ JSD(P||Q) = \frac{1}{2} (1 \log \frac{1}{1} + \cdots 0 + \cdots 0) = 0 \]

\[ \theta = 0 \]

at \( x = 0 \), \quad m = \frac{1 + 0}{2} = 0.5

\[ JSD(P||Q) = \frac{1}{2} \left( 1 \log \frac{1}{0.5} + 1 \log \frac{1}{0.5} + \cdots 0 + \cdots 0 \right) \]

\[ = \log (2) \]
Difficulty in Training GANs

Troubles with JS Divergence

The Gradients are unable to tell the model that it should move from the first case to the second.

\[ \text{JS Divergence is nearly identical either way} \]
\[ \rightarrow \text{ The overlap is nearly 0.} \]

The Gradients are unable to tell the model that it should move from the first case to the second.
Improving GANs

A better distance metric than JSD

What we want: A metric to tell us how far apart the two distributions are, but not just based on probabilities.
Improving GANs
A better distance metric than JSD

Imagine all the little points to be specs of dirt, and you want to move one pile of dirt over to another.

Earth mover’s Distance or Optimal Transport:
How much distance you need to cover to move all the parts of one distribution to the other?
**Wasserstein GAN**

**Formulation**

**Earth mover’s Distance or Optimal Transport:**
How much distance you need to cover to move all the parts of one distribution to the other?

\[
W(P_X, P_G) = \inf_{\gamma \in \Gamma(P_X, P_G)} E_{(x,y) \sim \gamma(x,y)} \left[ ||x - y|| \right]
\]

- Mapping of which x goes to which y
- Distance between the points

**Inf** (Topic for real analysis):
Infimum, the closest thing to a lower bound you can get.
As far as we are concerned, an alternative to min.

**Goal:** Find the optimal mapping, or optimal plan, of moving the specs of dirt from one place to another.
Earth mover’s Distance or Optimal Transport:
How much distance you need to cover to move all the parts of one distribution to the other?

\[ W(P_X, P_G) = \inf_{\gamma \in \Gamma(P_X, P_G)} \mathbb{E}_{(x, y) \sim \gamma}(x, y)[||x - y||] \]

But...

1. We don’t know what \( P_X \) is
2. \( \gamma \) could be super complex itself

Which make learning \( \gamma \) directly a very difficult problem.

This is where the Kantorovich-Rubinstein duality comes in:

\[ W(P_X, P_G) = \sup_{||f||_{L^1} \leq 1} \mathbb{E}_{P_X}[f(x)] - \mathbb{E}_{P_G}[f(x)] \]

Which looks a lot like the GANs we are familiar with!

\[ \min_{G} \max_{D} \mathbb{E}_{P_X} \log D(x) - \mathbb{E}_{P_Z} \log(D(G(z))) \]

Arjovsky et al, 2017, Wasserstein GAN
**Wasserstein GAN**

**Formulation**

**Earth mover’s Distance** or **Optimal Transport:**
How much distance you need to cover to move all the parts of one distribution to the other?

\[
W(P_X, P_G) = \inf_{\gamma \in \Gamma(P_X, P_G)} \mathbb{E}_{(x,y) \sim \gamma(x,y)}[||x - y||]
\]

But...

1. We don’t know what \(P_X\) is
2. \(\gamma\) could be super complex itself

Which make learning \(\gamma\) directly a very difficult problem.

This is where the Kantorovich-Rubinstein duality comes in:

\[
W(P_X, P_G) = \sup_{\|f\|_L \leq 1} \mathbb{E}_{P_X}[f(x)] - \mathbb{E}_{P_G}[f(x)]
\]

Which looks a lot like the GANs we are familiar with!

\[
\min_G \max_D \mathbb{E}_{P_X} \log D(x) - \mathbb{E}_{P_Z} \log(D(G(z)))
\]

Arjovsky et al, 2017, Wasserstein GAN
Earth mover’s Distance or Optimal Transport:  How much distance you need to cover to move all the parts of one distribution to the other?

\[ W(P_X, P_G) = \sup_{||f||_1 \leq 1} \mathbb{E}_{P_X}[f(x)] - \mathbb{E}_{P_G}[f(x)] \]

How?
One way (not the best way) is to clip Weights:

1. One layer: \( f(x; \theta) = ReLU(W_1 x + b_1) \)
   if \( W_{1,i,j} \in [-0.1, 0.1] \), then the slope can’t be more than 0.1 (times the dimensionality)

2. Two layer: \( f(x; \theta) = W_2 ReLU(W_1 x + b_1) + b_2 \)
   if \( W_{1/2,i,j} \in [-0.1, 0.1] \), then the slope is still bounded (not 0.1 though)

We can’t guarantee the function to be 1-Lipschitz, but some K-Lipschitz for some finite K.
And that’s enough.
Wasserstein GAN

Formulation

Earth mover’s Distance or Optimal Transport:
How much distance you need to cover to move all the parts of one distribution to the other?

\[ W(P_X, P_G) = \sup_{\|f\|_L \leq 1} \mathbb{E}_{P_X}[f(x)] - \mathbb{E}_{P_G}[f(x)] \]

How?

1. Update \( D(\theta_D) \) using gradient of \( \mathbb{E}_{x \sim P_X}[D(x)] - \mathbb{E}_{z \sim P_Z}[D(G(z))] \)
   (we interchange \( z \) and \( x \) in different equations for convenience, but they mean the same thing)

2. Clip all the weights in \( \theta_D \) to some \([-c, c]\)

3. Update \( G(\theta_G) \) to minimize \( -\mathbb{E}_{z \sim P_Z}[D(G(z))] \)
   (or equivalently maximize \( \mathbb{E}_{z \sim P_Z}[D(G(z))] \) )

Arjovsky et al, 2017, Wasserstein GAN
Wasserstein GAN

Graphics

Figure 2: Optimal discriminator and critic when learning to differentiate two Gaussians. As we can see, the discriminator of a minimax GAN saturates and results in vanishing gradients. Our WGAN critic provides very clean gradients on all parts of the space.
Wasserstein GAN
Formulation

Earth mover’s Distance or Optimal Transport:
How much distance you need to cover to move all the parts of one distribution to the other?

\[ W(P_X, P_G) = \sup_{\|f\|_L \leq 1} \mathbb{E}_P[f(x)] - \mathbb{E}_G[f(x)] \]

How?
1. Update \( D(\theta_D) \) using gradient of \( \mathbb{E}_{x \sim P_X}[D(x)] - \mathbb{E}_{z \sim P_Z}[D(G(z))] \)
   (we interchange z and x in different equations for convenience, but they mean the same thing)
2. Clip all the weights in \( \theta_D \) to some \([-c, c]\)
3. Update \( G(\theta_G) \) to minimize \(-\mathbb{E}_{z \sim P_Z}[D(G(z))]\)
   (or equivalently maximize \( \mathbb{E}_{z \sim P_Z}[D(G(z))] \))

But
If the clipping parameter is too large, then it takes too long to train the Discriminator to optimality, 
If too small, then the gradients vanish before reaching the generator.

Not the best way to do this..

Arjovsky et al, 2017, Wasserstein GAN
Wasserstein GAN

Gradient Penalty

**Earth mover’s Distance** or **Optimal Transport:**
How much distance you need to cover to move all the parts of one distribution to the other?

$$W(P_X, P_G) = \sup_{||f||_{L^1} \leq 1} \mathbb{E}_{P_X}[f(x)] - \mathbb{E}_{P_G}[f(x)]$$

**What if:** we instead directly attempt to include the Lipschitz constraint in the equation?

$$\mathbb{E}_{x \sim P_X} \left[ D(x) - \lambda \left( ||\nabla_x D(x)||_2 - 1 \right)^2 \right] - \mathbb{E}_{z \sim P_Z}[D(G(z))]$$

**Basically:** Push the norm of the gradient to be close to 1

Gulrajani et al, 2017, Improved Training of Wasserstein GANs
Earth mover’s Distance or Optimal Transport:
How much distance you need to cover to move all the parts of one distribution to the other?

\[
W(P_X, P_G) = \sup_{\|f\|_L \leq 1} \mathbb{E}_P[f(x)] - \mathbb{E}_G[f(x)]
\]

\[
W_l \leftarrow \frac{W_l}{\sigma(W_l)}
\]

Where \(\sigma(W_l)\) is the spectral norm of \(W_l\)

We won’t go into details here, but it is sufficient to know that this is a more “principled” way to enforce the Lipschitz constraint on the weights.

It comes from the fact that the Lipschitz norm (the bound) for a composite function is upper bounded by the product of the Lipschitz norms of the composing functions.

Miyato et al, 2018, Spectral Normalization for Generative Adversarial Networks
Poll (@1704)

Which of the following is True:

A. KL Divergence can avoid exploding gradient problem.
B. Wasserstein Distance can fix vanishing gradient problem
C. JS Divergence can fix vanishing gradient problem
D. JS Divergence can avoid exploding gradient problem.
Poll (@1704)

Which of the following is True:

A. KL Divergence can avoid exploding gradient problem.
B. Wasserstein Distance can fix vanishing gradient problem.
C. JS Divergence can fix vanishing gradient problem.
D. JS Divergence can avoid exploding gradient problem.
With the advent of WGANs and some other clever ways to give us easier training for GANs, there were several types of GANs proposed in literature:

1. Conditional GAN
2. LapGAN (Laplacian GAN, stacked Conditional GANs)
3. BiGAN (Involves both $z \rightarrow \hat{x}$ and $\hat{x} \rightarrow z$ )
4. Recurrent Adversarial Network
5. Categorical GAN
6. InfoGAN
7. AAE
8. STAR-GAN
9. Pix2Pix
10. CycleGAN
Conditional GAN
What is it?

- Provides more control in what type of images are generated by the generator.
- Are not strictly unsupervised learning algorithms because they require labeled data as input to the additional layer.
- Allows us to generate the outputs for exactly the classes or labels that we desire.
Conditional GAN
Injecting extra information

Mathematical expression for a simple GAN -

\[
\min_G \max_D V(D, G) = \mathbb{E}_{x \sim p_{data}(x)} [\log D(x)] + \mathbb{E}_{z \sim p_z(z)} [\log(1 - D(G(z)))]
\]

Mathematical expression for a Conditional GAN -

\[
\min_G \max_D V(D, G) = \mathbb{E}_{x \sim p_{data}(x)} [\log D(x|y)] + \mathbb{E}_{z \sim p_z(z)} [\log(1 - D(G(z|y)))]
\]
Conditional GAN

Applications

Image-to-image translation
Conditional GAN

Applications

This bird is white with some black on its head and wings, and has a long orange beak

This bird has a yellow belly and tarsus, grey back, wings, and brown throat, nape with a black face

This flower has overlapping pink pointed petals surrounding a ring of short yellow filaments

(a) StackGAN
Stage-I
64x64 images

(b) StackGAN
Stage-II
256x256 images

(c) Vanilla GAN
256x256 images

Text to image synthesis

Figure 1. Comparison of the proposed StackGAN and a vanilla one-stage GAN for generating 256×256 images. (a) Given text descriptions, Stage-I of StackGAN sketches rough shapes and basic colors of objects, yielding low-resolution images. (b) Stage-II of StackGAN takes Stage-I results and text descriptions as inputs, and generates high-resolution images with photo-realistic details. (c) Results by a vanilla 256×256 GAN which simply adds more upsampling layers to state-of-the-art GAN-INT-CLS [26]. It is unable to generate any plausible images of 256×256 resolution.
GANs can be used for much more than image generation. By clever training, we can even use them for image translation.

Zhu et al, 2017, *Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks*
Why CycleGAN?

Let's try methods we have discussed so far
Cycle GAN
Autoencoder Alternative

Unless you have paired images, achieving the above with a simple Autoencoder is simply not possible.

So, what if we just threw a GAN (or a WGAN) at it?

Zhu et al, 2017, Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks
This only trains the GAN to produce Zebra-like images. Not necessarily having to do ANYTHING with the input image.

In short, this may not translate but instead completely ignore what you fed in.

We also need a way to make sure that the produced image is closely related to the input image for the generator.

Zhu et al, 2017, Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks
This is where **Cycle GANs** come in. On top of the regular GAN training to ensure that the generated image looks like a zebra, the cycle consistency loss (just a reconstruction loss) ensures that the generated image has retained enough information about the input enough to retrieve it using a different “special purpose” generator.

Zhu et al, 2017, *Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks*
What we learn from CycleGAN

• There is no paired dataset of Zebras and Horses
• So there is no easy discriminative method to train Zebras from Horses
• But using GANs, we can train distributions to match.
Neural Style Transfer Results
Style transfer with CycleGAN
Comparing

• **Neural Style Transfer**
  - Need a content and style image
  - Specific & small number of images
  - More control
  - [https://reinakano.com/arbitrary-image-stylization-tfjs/](https://reinakano.com/arbitrary-image-stylization-tfjs/)

• **CycleGAN**
  - Just need 2 domains of images. No need for specific content or style images
  - Many similar pictures
  - Specificity of images doesn’t really matter
Comparing

• Neural Style Transfer
  - Need a content and style image
  - Specific & small number of images
  - More control
  - https://reinakano.com/arbitrary-image-stylization-tfjs/

• CycleGAN
  - Just need 2 domains of images. No need for specific content or style images
  - Many similar pictures
  - Specificity of images doesn’t really matter
Comparing

- **Neural Style Transfer**
  - Need a content and style image
  - Specific & small number of images
  - More control
  - [https://reinakano.com/arbitrary-image-stylization-tfjs/](https://reinakano.com/arbitrary-image-stylization-tfjs/)

- **CycleGAN**
  - Just need 2 domains of images. No need for specific content or style images
  - Many similar pictures
  - Specificity of images doesn’t really matter
StarGAN
Generalization of CycleGAN

Given training data from two different domains, StarGANs learn to translate images from one domain to the other. For example — changing the hair color (attribute) of a person from black (attribute value) to blond (attribute value).

StarGAN

How does it work?

- G takes in as input both the image and target domain label and generates a fake image.
- It then tries to reconstruct the original image from the fake image conditioned on the original domain label.
- Discriminator tells if image is fake and classifies an image to its corresponding domain, so G will ultimately learn to generate realistic images corresponding to the given target domain. (d)
StarGAN
Image-to-Image Translation

(a) Training the discriminator

Real image

Fake image

(b) Original-to-target domain

Output image and original domain label

(c) Target-to-original domain

(d) Fooling the discriminator

Input image and target domain label

Reconstructed image

Training with CelebA

Deep Learning Notes: StarGAN. From the perceived daunting task of... | by ashwin bhat | Medium
Objectives of Discriminator:

1. Identify whether an image is fake or not.
2. Predict the target domain of the input image.
   ○ Uses an auxiliary classifier to learn the mapping of original image and its corresponding domain from the dataset.
Objectives of the Generator:

1. The images generated are realistic.
2. The images generated are classifiable as target domain by D.
3. Able to reconstruct the original image from the fake image given the original domain label. (Cycle consistency Loss)
BiGAN
Adversarial Feature Learning

A Bi-Directional Generative Adversarial Network trains an encoder/decoder pair in an elegant fashion. The discriminator tries to tell the difference between pairs of real data and encoded real data from data generated from prior samples and prior samples.

[DKD16]

The BiGAN training objective is defined as a minimax objective

$$\min_{G} \max_{D} V(D, E, G)$$

(2)

where

$$V(D, E, G) := \mathbb{E}_{x \sim p_{x}} \left[ \mathbb{E}_{z \sim p_{E}(\cdot|x)} \left[ \log D(x, z) \right] \right] + \mathbb{E}_{z \sim p_{z}} \left[ \mathbb{E}_{x \sim p_{G}(\cdot|z)} \left[ \log (1 - D(x, z)) \right] \right].$$

(3)
BiGAN

Adversarial Feature Learning

Donahue et al, 2016, Adversarial Feature Learning
A Laplacian GAN is constructed of a chain of conditional GANs, to generate progressively larger images. A GAN generates small, blurry images. A conditional GAN generates larger images conditioned on the smaller image, repeated until you reach the desired size.[DCSF15]
Figure 2: The training procedure for our LAGPAN model. Starting with a 64x64 input image \( I \) from our training set (top left): (i) we take \( I_0 = I \) and blur and downsample it by a factor of two (red arrow) to produce \( I_1 \); (ii) we upsample \( I_1 \) by a factor of two (green arrow), giving a low-pass version \( l_0 \) of \( I_0 \); (iii) with equal probability we use \( l_0 \) to create either a real or a generated example for the discriminative model \( D_0 \). In the real case (blue arrows), we compute high-pass \( h_0 = I_0 - l_0 \) which is input to \( D_0 \) that computes the probability of it being real vs generated. In the generated case (magenta arrows), the generative network \( G_0 \) receives as input a random noise vector \( z_0 \) and \( l_0 \). It outputs a generated high-pass image \( \tilde{h}_0 = G_0(z_0, l_0) \), which is input to \( D_0 \). In both the real/generated cases, \( D_0 \) also receives \( l_0 \) (orange arrow). Optimizing Eqn. 2, \( G_0 \) thus learns to generate realistic high-frequency structure \( \tilde{h}_0 \) consistent with the low-pass image \( l_0 \). The same procedure is repeated at scales 1 and 2, using \( I_1 \) and \( I_2 \). Note that the models at each level are trained independently. At level 3, \( I_3 \) is an 8x8 image, simple enough to be modeled directly with a standard GANs \( G_3 \) & \( D_3 \).
GANs
And Many More…
Summary

• Training GANs is difficult for a lot of reasons
• Primarily, the issues of the discriminator capturing the wrong boundary, being too confident (and non differentiable), and getting saturated due to the formulation of the objective.
• We can remedy this by using Earth mover’s metric – Wasserstein Distance
• The Lipschitz constraint of WGANs can be imposed in several ways including weight clipping, gradient penalties, and spectral norm.
• There are (so) many variants of GANs which do more than just image generation and are able to do image translation (or style transfer) as well!
References

• [StarGAN — Image-to-Image Translation | by Pranoy Radhakrishnan | Towards Data Science](https://towardsdatascience.com/)
• [Deep Learning Notes: StarGAN. From the perceived daunting task of... | by ashwin bhat | Medium](https://medium.com/)
