Generative Adversarial Networks

Vijay Raghavan

Agenda

1. Introduction
2. Wasserstein Generative Adversarial Network (WGAN)
3. Optimizations for WGANs
4. Conditional Generative Adversarial Network (CGAN)
5. CycleGAN
6. StyleGAN
7. Visual Paragraph Generation

Introduction to Generative Adversarial Networks

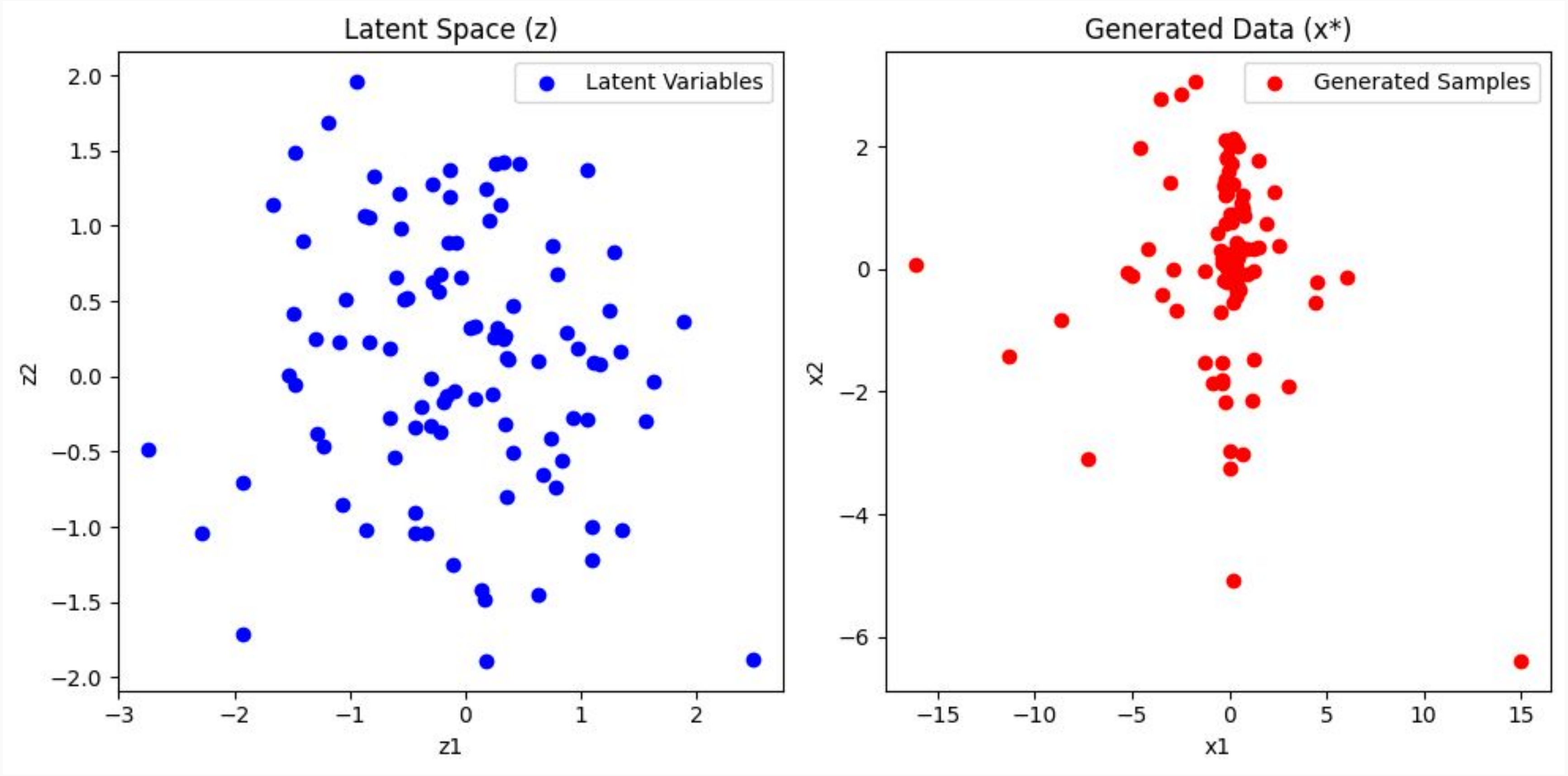
Overview

- In a GAN, the main generator network creates samples by mapping random noise to the output data space.
- If a second discriminator network cannot distinguish between the generated samples and the real examples, the samples must be plausible.
- If this network can tell the difference, this provides a training signal that can be fed back to improve the quality of the samples.

Generator

- We aim to generate new samples $\{\boldsymbol{x}_j^*\}$ that are drawn from the same distribution as a **j** set of real training data $\{\boldsymbol{x}_i\}$.
- A single new sample $\boldsymbol{x}_j^*$ is generated by
  - (i) choosing a latent variable $\boldsymbol{z}_j$ from a simple base distribution (e.g., a standard normal) and then
  - (ii) passing this data through a network $\boldsymbol{x^* = g\,[z_j, \theta]}$ with parameters $\boldsymbol{\theta}$.
- This network is known as the generator.
- During the learning process, the goal is to find parameters $\boldsymbol{\theta}$ so hat the samples $\{\boldsymbol{x}_j^*\}$ look "similar" to the real data $\{\boldsymbol{x}_i\}$

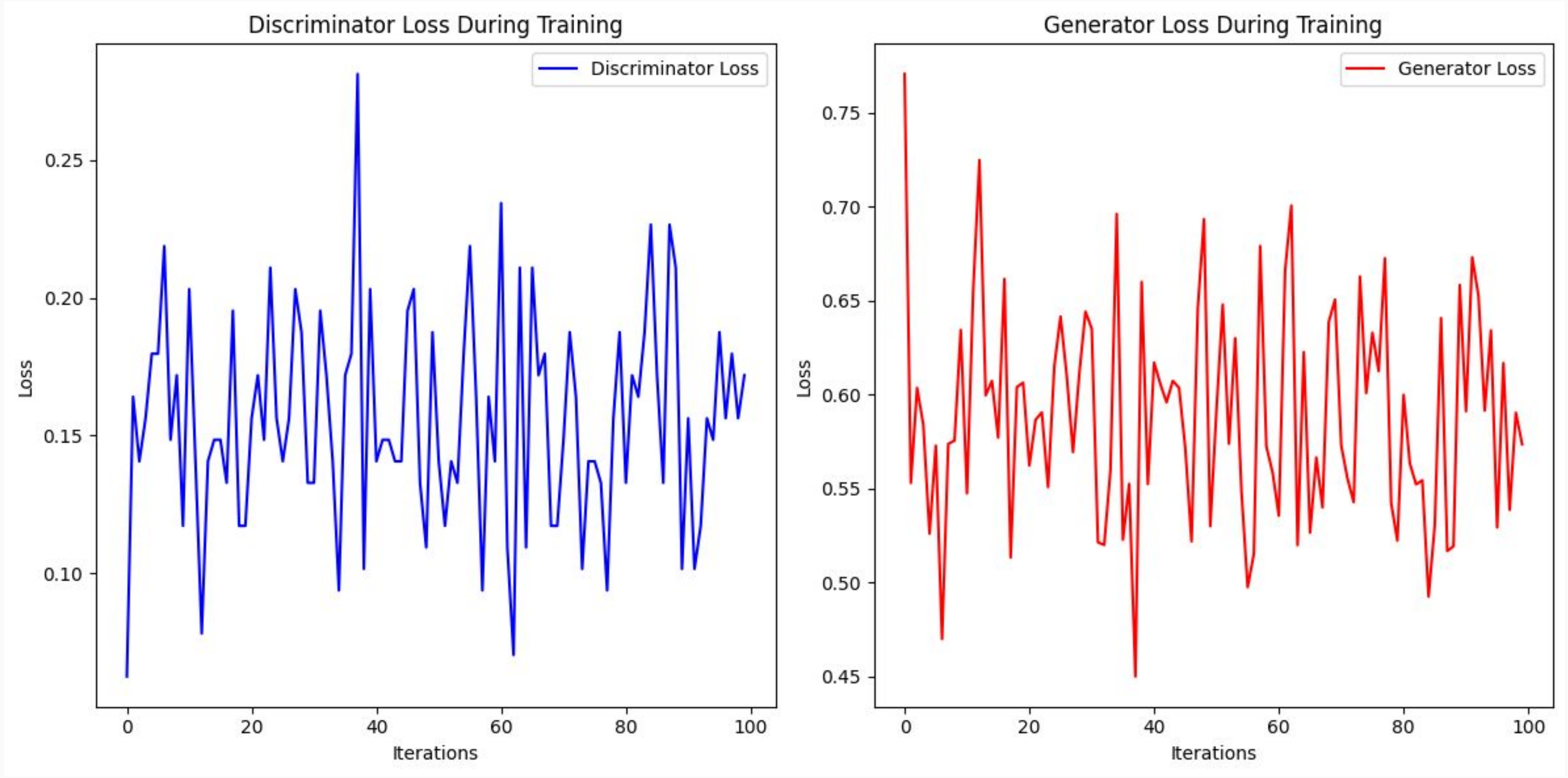
Generator transforms new latent variables into generated samples



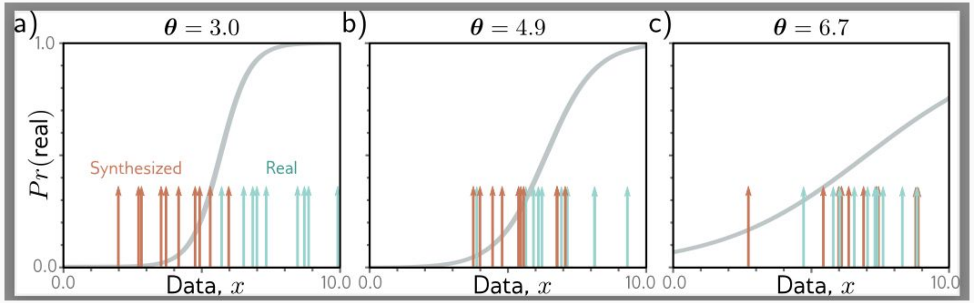
Discriminator

- Similarity can be defined in many ways, but GAN uses the principle that the samples should be statistically indistinguishable from the true data.
- To this end, a second network $\boldsymbol{f[\bullet,\varphi]}$ with parameters $\boldsymbol{\varphi}$ called the discriminator is introduced.
- This network aims to classify its input as being a real example or a generated sample.
- If proves impossible, the generated samples are indistinguishable from the real examples, and we have succeeded. If it is possible, the discriminator provides a signal that can be used to improve the generation process.

Adversarial training process



GAN Mechanism



a) Given a parameterized function (a generator) that synthesizes samples (orange arrows) and a batch of real examples (cyan arrows), we train a discriminator to distinguish the real examples from the generated samples (sigmoid curve indicates the estimated probability that the data point is real).

b) The generator is trained by modifying its parameters so that the discriminator becomes less confident the samples were synthetic (in this case, by moving the orange samples to the right). The discriminator is then updated.

c) Alternating updates to the generator and discriminator cause the generated samples to become indistinguishable from real examples and the impetus to change the generator (i.e., the slope of the sigmoid function) to diminish.

Prince, S. J. (2023). *Understanding Deep Learning*. MIT press.

GAN Loss Functions

- The discriminator ***f[x, φ]*** takes input ***x***, has parameters ***φ***, and returns a scalar that is higher when it believes the input is a real example.
- This is a binary classification task, so we adapt the binary cross-entropy loss function as the starting point where ***y****$_i$* ***∈ {0, 1}*** is the label, and ***sig [•]*** is the logistic sigmoid function.

$$\hat{\phi} = \underset{\phi}{\operatorname{argmin}} \left[\sum_i -(1-y_i)\log\left[1 - \mathrm{sig}[\mathrm{f}[\mathbf{x}_i, \phi]]\right] - y_i \log\left[\mathrm{sig}[\mathrm{f}[\mathbf{x}_i, \phi]]\right] \right]$$

- In this case, we assume that the real examples ***x*** have label ***y = 1*** and the generated samples ***x****** have label ***y = 0*** so that:

$$\hat{\phi} = \underset{\phi}{\operatorname{argmin}} \left[\sum_j -\log\left[1 - \mathrm{sig}[\mathrm{f}[\mathbf{x}^*_j, \phi]]\right] - \sum_i \log\left[\mathrm{sig}[\mathrm{f}[\mathbf{x}_i, \phi]]\right] \right]$$

Where i and j index the real examples and generated samples,
Now we substitute the definition for the generator ***x****$^*_j$* ***= g [z****$_j$***,θ]*** and note that we must maximize with respect to ***θ*** since we want the generated samples to be misclassified (i.e., have low likelihood of being synthetic or high negative log-likelihood):

$$\hat{\theta} = \underset{\theta}{\operatorname{argmax}} \left[\min_{\phi} \left[\sum_j -\log\left[1 - \mathrm{sig}[\mathrm{f}[\mathbf{g}[\mathbf{z}_j, \theta], \phi]]\right] - \sum_i \log\left[\mathrm{sig}[\mathrm{f}[\mathbf{x}_i, \phi]]\right] \right] \right]$$

Prince, S. J. (2023). *Understanding Deep Learning*. MIT press.

Analysis of the Loss function

- If we divide the two sums in the previous equation by the numbers *I,J* of real and generated samples, then the loss function can be written in terms of expectations
- and *P*$_r$*(x*)* *is* the probability distribution over the generated samples, and *P*$_r$*(x)* is the true probability distribution over the real examples.

$$\begin{aligned} L[\phi] &= \frac{1}{J}\sum_{j=1}^{J}\left(\log\left[1-\text{sig}[\text{f}[\mathbf{x}_j^*,\phi]]\right]\right)+\frac{1}{I}\sum_{i=1}^{I}\left(\log\left[\text{sig}[\text{f}[\mathbf{x}_i,\phi]]\right]\right) \\ &\approx \mathbb{E}_{\mathbf{x}^*}\left[\log\left[1-\text{sig}[\text{f}[\mathbf{x}^*,\phi]]\right]\right]+\mathbb{E}_{\mathbf{x}}\left[\log\left[\text{sig}[\text{f}[\mathbf{x},\phi]]\right]\right] \\ &= \int Pr(\mathbf{x}^*)\log\left[1-\text{sig}[\text{f}[\mathbf{x}^*,\phi]]\right]d\mathbf{x}^*+\int Pr(\mathbf{x})\log\left[\text{sig}[\text{f}[\mathbf{x},\phi]]\right]d\mathbf{x}, \end{aligned}$$

The optimal discriminator for an example *x̃* of unknown origin is:

$$Pr(\text{real}|\tilde{\mathbf{x}}) = \text{sig}\left[\text{f}[\tilde{\mathbf{x}},\phi]\right] = \frac{Pr(\tilde{\mathbf{x}}|\text{real})}{Pr(\tilde{\mathbf{x}}|\text{generated})+Pr(\tilde{\mathbf{x}}|\text{real})} = \frac{Pr(\mathbf{x})}{Pr(\mathbf{x}^*)+Pr(\mathbf{x})},$$

Prince, S. J. (2023). *Understanding Deep Learning*. MIT press.

Quality and Coverage

Substituting the probabilities in the loss function we get:

$$\begin{aligned} L[\phi] &= \int Pr(\mathbf{x}^*)\log\Big[1-\text{sig}[\text{f}[\mathbf{x}^*,\phi]]\Big]d\mathbf{x}^* + \int Pr(\mathbf{x})\log\Big[\text{sig}[\text{f}[\mathbf{x},\phi]]\Big]d\mathbf{x} \qquad (15.8) \\ &= \int Pr(\mathbf{x}^*)\log\left[1-\frac{Pr(\mathbf{x})}{Pr(\mathbf{x}^*)+Pr(\mathbf{x})}\right]d\mathbf{x}^* + \int Pr(\mathbf{x})\log\left[\frac{Pr(\mathbf{x})}{Pr(\mathbf{x}^*)+Pr(\mathbf{x})}\right]d\mathbf{x} \\ &= \int Pr(\mathbf{x}^*)\log\left[\frac{Pr(\mathbf{x}^*)}{Pr(\mathbf{x}^*)+Pr(\mathbf{x})}\right]d\mathbf{x}^* + \int Pr(\mathbf{x})\log\left[\frac{Pr(\mathbf{x})}{Pr(\mathbf{x}^*)+Pr(\mathbf{x})}\right]d\mathbf{x}. \end{aligned}$$

Disregarding additive and multiplicative constants, this is the Jensen-Shannon divergence between the synthesized distribution Pr(x^*) and the true distribution Pr(x) and where $D_{KL}[\bullet||\bullet]$ is the Kullback-Leibler divergence

$$\begin{aligned} &D_{JS}\Big[Pr(\mathbf{x}^*)\,||\,Pr(\mathbf{x})\Big] \qquad (15.9) \\ &\quad= \frac{1}{2}D_{KL}\left[Pr(\mathbf{x}^*)\,\middle|\middle|\,\frac{Pr(\mathbf{x}^*)+Pr(\mathbf{x})}{2}\right] + \frac{1}{2}D_{KL}\left[Pr(\mathbf{x})\,\middle|\middle|\,\frac{Pr(\mathbf{x}^*)+Pr(\mathbf{x})}{2}\right] \\ &\quad= \underbrace{\frac{1}{2}\int Pr(\mathbf{x}^*)\log\left[\frac{2Pr(\mathbf{x}^*)}{Pr(\mathbf{x}^*)+Pr(\mathbf{x})}\right]d\mathbf{x}^*}_{\text{quality}} + \underbrace{\frac{1}{2}\int Pr(\mathbf{x})\log\left[\frac{2Pr(\mathbf{x})}{Pr(\mathbf{x}^*)+Pr(\mathbf{x})}\right]d\mathbf{x}}_{\text{coverage}}. \end{aligned}$$

- The first term penalizes regions with samples x^* but no real examples x; it enforces quality.
- The second penalizes regions with real examples but no samples. It enforces coverage.

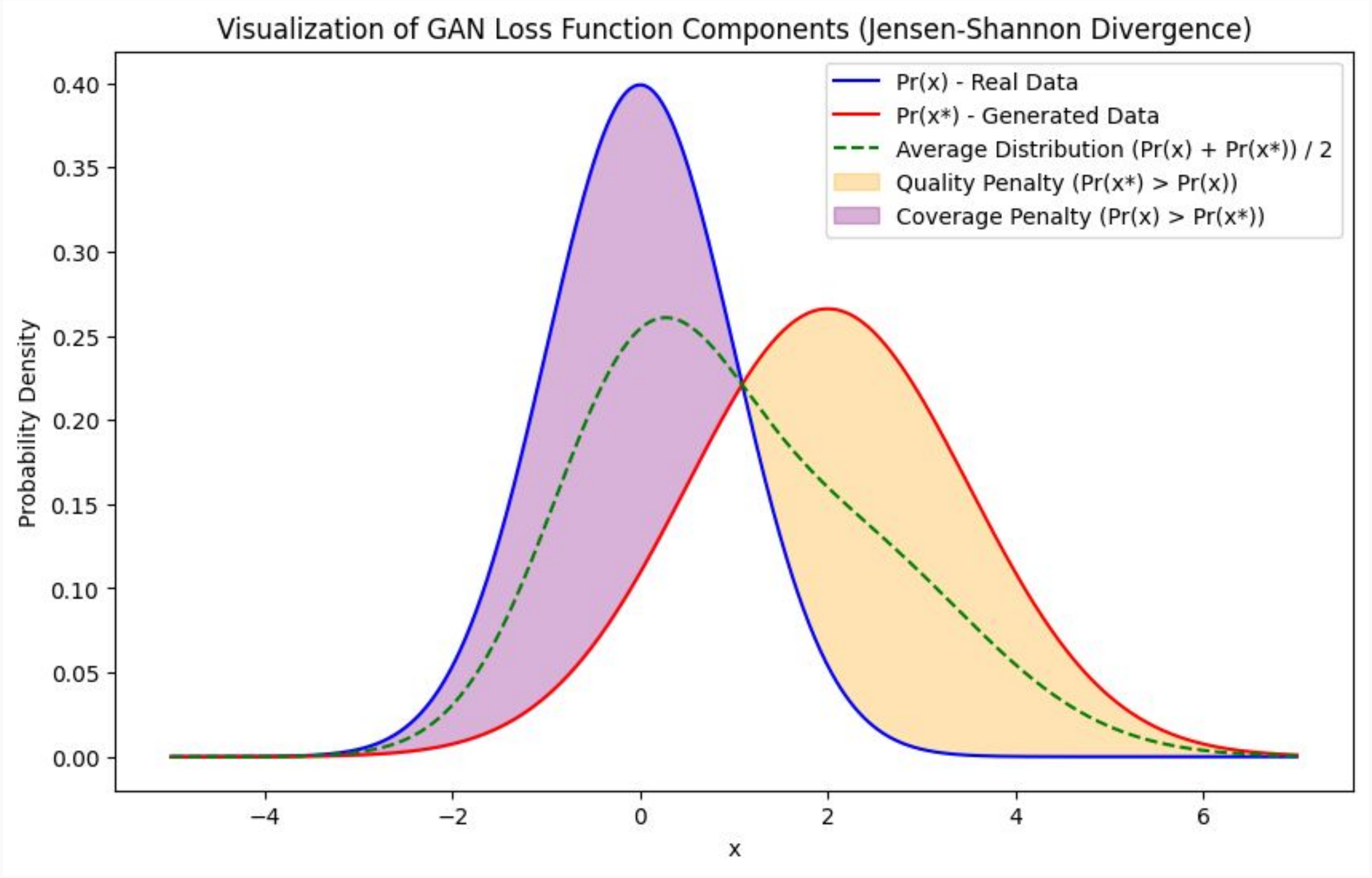
Prince, S. J. (2023). *Understanding Deep Learning*. MIT press.

GAN Loss Function Analysis

- **Jensen-Shannon Divergence:**
  - Average of two Kullback-Leibler (KL) divergences
  - First KL term: "Quality" - penalizes when generated density high but mixture density low
  - Second KL term: "Coverage" - penalizes when true density high but mixture density low
- **Limitation:**
  - Second "Coverage" term independent of generator
  - Generator optimizes for quality but not explicit coverage
  - Hypothesized reason for mode collapse/dropping in GANs
- **Issue:**
  - If generated samples easily distinguishable from real
  - Discriminator sigmoid has shallow slope at sample positions
  - Tiny gradients to update generator, hindering full distribution capture

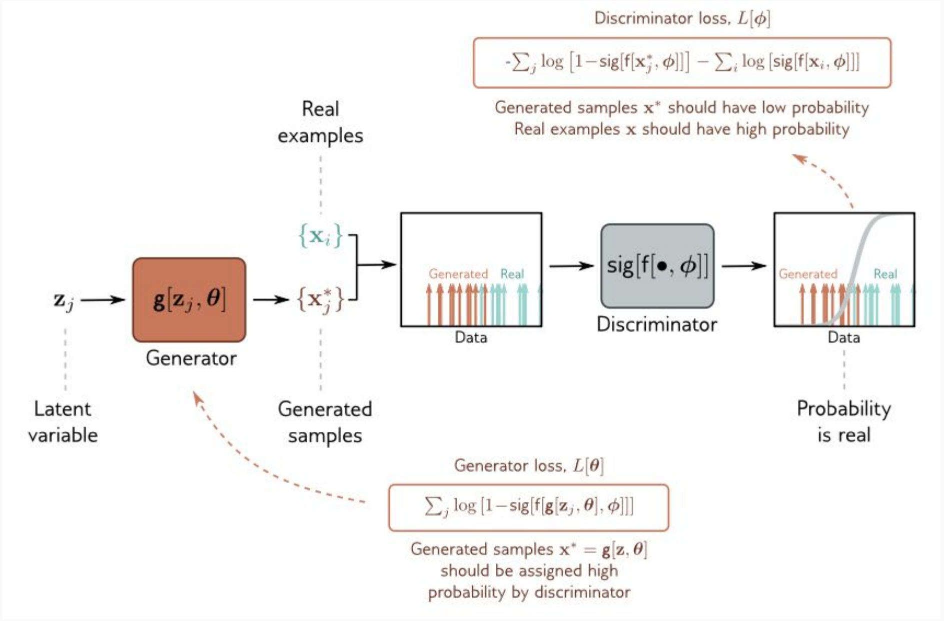
TL;DR: GAN loss function encourages quality but lacks explicit coverage term, potentially contributing to mode collapse.

Quality and Coverage Penalty



GAN Training

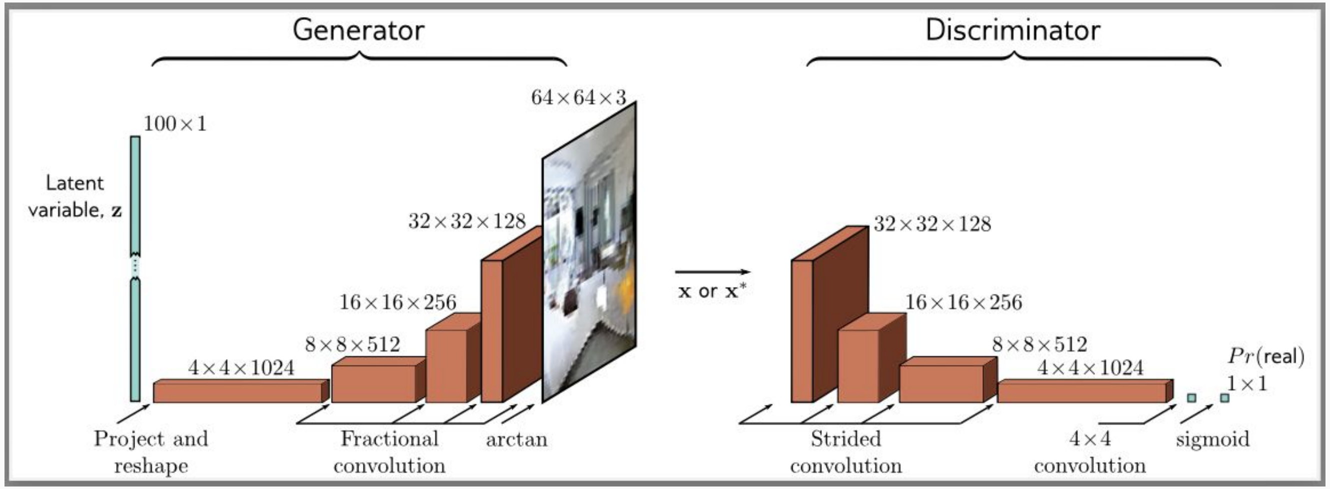
- A latent variable z_j is drawn from the base distribution and passed through the generator to create a sample x^*.
- A batch $\{x^*_j\}$ of samples and a batch of real examples $\{x_i\}$ are passed to the discriminator, which assigns a probability that each is real.
- The discriminator parameters φ are modified to assign high probability to the real examples and low probability to the generated samples.
- The generator parameters θ are modified to "fool" the discriminator into assigning the generated samples a high probability.



Discriminator loss, $L[\phi]$
$-\sum_j \log\left[1-\text{sig}[\text{f}[\mathbf{x}^*_j,\phi]]\right] - \sum_i \log\left[\text{sig}[\text{f}[\mathbf{x}_i,\phi]]\right]$
Generated samples $\mathbf{x}^*$ should have low probability
Real examples $\mathbf{x}$ should have high probability
Real examples
$\{\mathbf{x}_i\}$
$\mathbf{z}_j$
$\mathbf{g}[\mathbf{z}_j,\boldsymbol{\theta}]$
Generator
$\{\mathbf{x}^*_j\}$
Generated
Real
Data
$\text{sig}[\text{f}[\bullet,\phi]]$
Discriminator
Generated
Real
Data
Latent variable
Generated samples
Probability is real
Generator loss, $L[\boldsymbol{\theta}]$
$\sum_j \log\left[1-\text{sig}[\text{f}[\mathbf{g}[\mathbf{z}_j,\boldsymbol{\theta}],\phi]]\right]$
Generated samples $\mathbf{x}^* = \mathbf{g}[\mathbf{z},\boldsymbol{\theta}]$ should be assigned high probability by discriminator

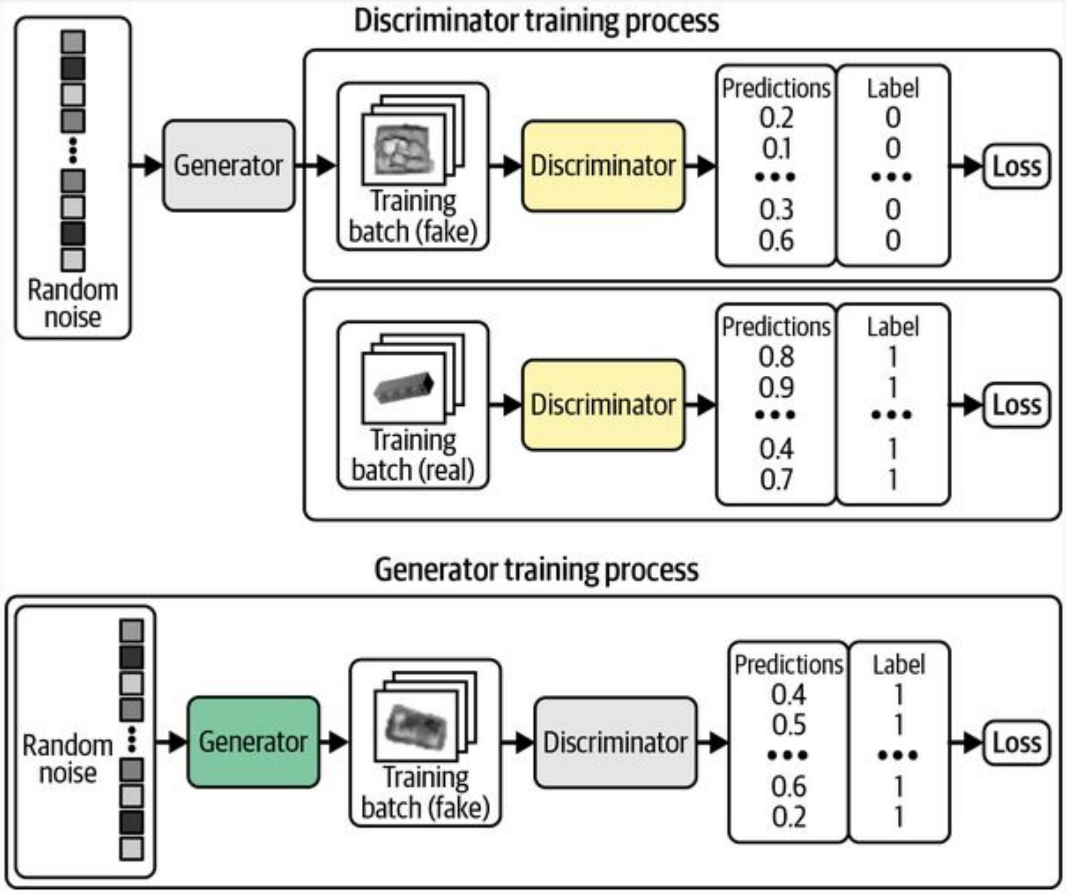
Prince, S. J. (2023). *Understanding Deep Learning*. MIT press.

Deep Convolutional GAN



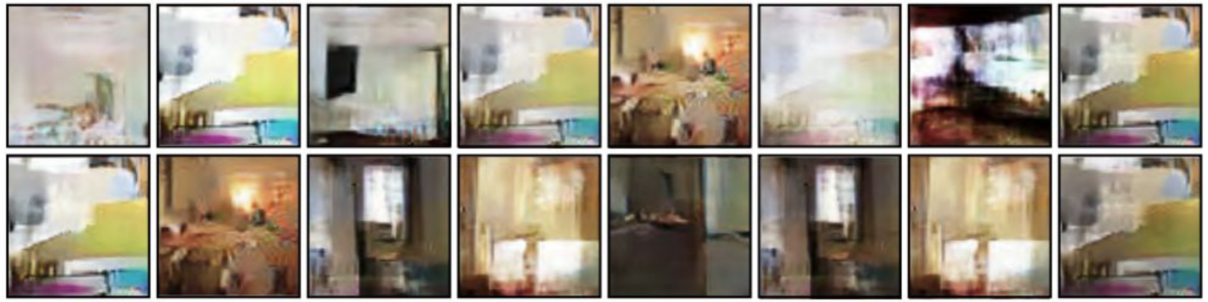
- In the generator, a 100D latent variable ***z*** is drawn from a uniform distribution and mapped by a linear transformation to a 4×4 representation with 1024 channels.
- This is then passed through a series of convolutional layers that gradually upsample the representation and decrease the number of channels.
- At the end is an arctan function that maps the 64×64×3 representation to a fixed range so that it can represent an image.
- The discriminator consists of a standard convolutional net that classifies the input as either a real example or a generated sample.

DCGAN


Discriminator training process
Random noise
Generator
Training batch (fake)
Discriminator
Predictions 0.2 0.1 ••• 0.3 0.6
Label 0 0 ••• 0 0
Loss
Training batch (real)
Discriminator
Predictions 0.8 0.9 ••• 0.4 0.7
Label 1 1 ••• 1 1
Loss
Generator training process
Random noise
Generator
Training batch (fake)
Discriminator
Predictions 0.4 0.5 ••• 0.6 0.2
Label 1 1 ••• 1 1
Loss

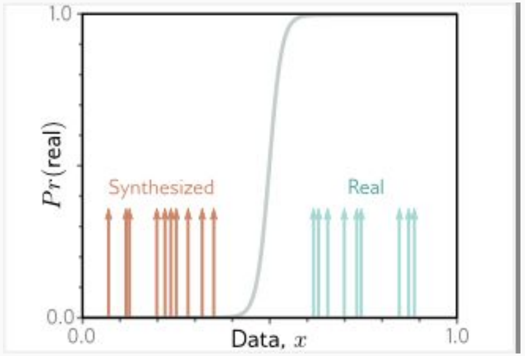
Mode Collapse

- A common failure mode is that the generator makes plausible samples, but these only represent a subset of the data (e.g., for faces, it might never generate faces with beards).
- This is known as mode dropping.
- An extreme version of this phenomenon can occur where the generator entirely or mostly ignores the latent variables ***z*** and collapses all samples to one or a few points; this is known as mode collapse



Vanishing gradients

- When the discriminator is optimal, the loss function minimizes a measure of the distance between the generated and real samples.
- If the probability distributions are completely disjoint, this distance is infinite, and any small change to the generator will not decrease the loss.
- The generated samples lie in a subspace that is the size of the latent variable z, and the real examples also lie in a low-dimensional subspace due to the physical processes that created the data .
- There may be little or no overlap between these subspaces, and the result is very small or no gradients
- If the discriminator can perfectly separate the generated and real samples, no small change to the generated data will change the classification score
- If the generated samples (orange arrows) are easy to distinguish from the real examples (cyan arrows),
  - then the discriminator (sigmoid) may have a very shallow slope at the positions of the samples; hence,
  - the gradient to update the parameter of the generator may be tiny.

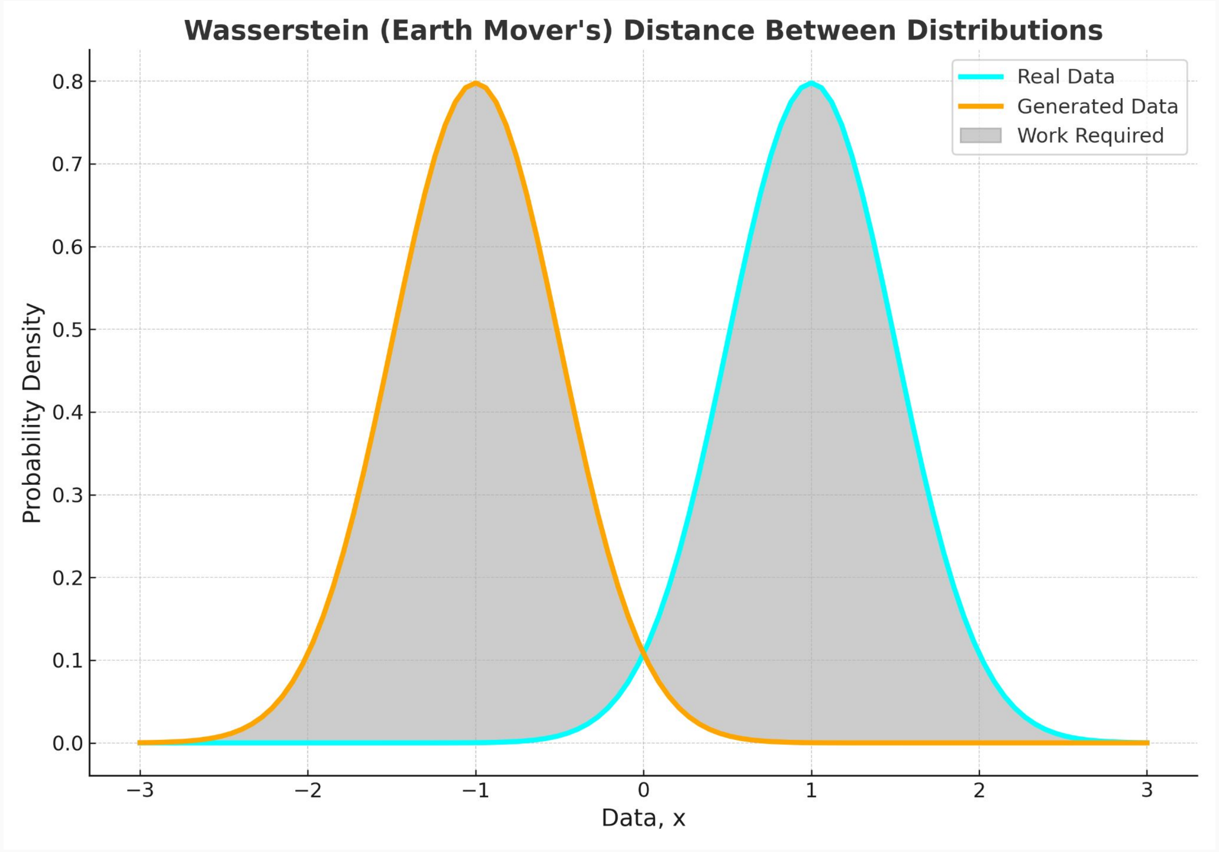


Wasserstein Generative Adversarial Network (WGAN)

Wasserstein distance

- GAN loss can be interpreted in terms of distances between probability distributions
- The gradient of this distance becomes zero when the generated samples are too easy to distinguish from the real examples.
- The obvious way forward is to choose a distance metric with better properties.
- The Wasserstein or (for discrete distributions) earth mover's distance is the quantity of work required to transport the probability mass from one distribution to create the other.
- "work" is defined as the mass multiplied by the distance moved.
- The Wasserstein distance is well-defined even when the distributions are disjoint and decreases smoothly as they become closer to one another.

Wasserstein (Earth Mover's) distance between the real and generated distributions



Wasserstein distance for discrete distributions

- Consider distributions Pr(x = i) and q(x = j) defined over K bins.
- Assume there is a cost C_{ij} associated with moving one unit of mass from bin i in the first distribution to bin j in the second this cost might be the absolute difference |i−j| between the indices.
- The amounts that are moved from the transport plan and are stored in a matrix P.
- The Wasserstein distance is defined as:

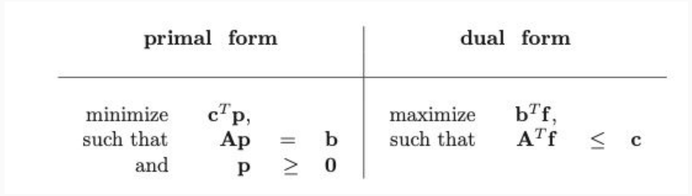
$$D_w\Big[Pr(x)||q(x)\Big] = \min_{\mathbf{P}} \left[\sum_{i,j} P_{ij} \cdot |i-j|\right],$$

subject to the constraints that:

$$\begin{aligned} \textstyle\sum_j P_{ij} &= Pr(x=i) && \text{initial distribution of } Pr(x) \\ \textstyle\sum_i P_{ij} &= q(x=j) && \text{initial distribution of } q(x) \\ P_{ij} &\geq 0 && \text{non-negative masses.} \end{aligned}$$

Constrained Optimization Problem

- This is a constrained minimization problem that maps the mass of one distribution to the other.
- We must solve this minimization problem over the elements Pij every time we want to compute the distance.



primal form				dual form			
minimize	$\mathbf{c}^T\mathbf{p}$,			maximize	$\mathbf{b}^T\mathbf{f}$,		
such that	$\mathbf{A}\mathbf{p}$	$=$	$\mathbf{b}$	such that	$\mathbf{A}^T\mathbf{f}$	$\leq$	$\mathbf{c}$
and	$\mathbf{p}$	$\geq$	$\mathbf{0}$				

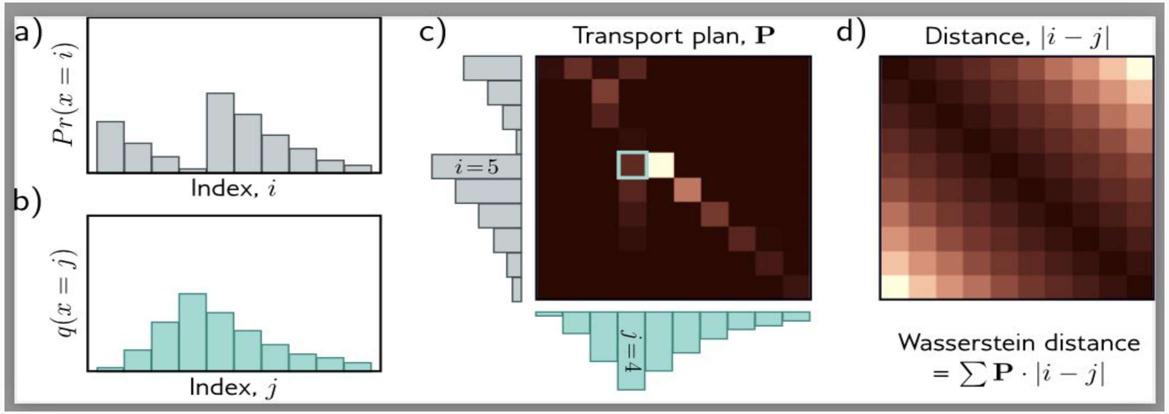
Where,

- p contains the vectorized elements Pij that determine the amount of mass moved,
- c contains the distances,
- Ap = b contains the initial distribution constraints, and p ≥ 0 ensures the masses moved are non-negative

We maximize with respect to a variable f that is applied to the initial distributions, subject to constraints that depend on the distances c

$$D_w\Big[Pr(x)||q(x)\Big] = \max_{\mathbf{f}}\left[\sum_i Pr(x=i)f_i - \sum_j q(x=j)f_j\right]$$

Computing Wasserstein Distance



- Consider the discrete distribution Pr(x = i) (a).
- We wish to move the probability mass to create the target distribution q(x = j) (b).
- The transport plan P identifies how much mass will be moved from i to j c).
- For example, the cyan highlighted square p_{54} indicates how much mass will be moved from i = 5 to j = 4.
- The elements of the transport plan must be non-negative, the sum over j must be Pr(x = i), and the sum over i must be q(x = j). Hence P is a joint probability distribution.
- The distance matrix between elements i and j (d).
- The optimal transport plan P minimizes the sum of the pointwise product of P and the distance matrix (termed the Wasserstein distance).
- Hence, the elements of P tend to lie close to the diagonal where the distance cost is lowest.

Wasserstein distance for continuous distributions

$$D_w\Big[Pr(\mathbf{x}), q(\mathbf{x})\Big] = \max_{\mathrm{f}[\mathbf{x}]}\left[\int Pr(\mathbf{x})\mathrm{f}[\mathbf{x}]d\mathbf{x} - \int q(\mathbf{x})\mathrm{f}[\mathbf{x}]d\mathbf{x}\right]$$

Subject to the constraint that the Lipschitz constant of the function f[x] is less than one (i.e., the absolute gradient of the function is less than one).

Wasserstein GAN loss function

In the context of neural networks, we maximize over the space of functions f[x] by optimizing the parameters φ in a neural network f[x, φ], and we approximate these integrals using generated samples x^*_i and real examples x_i:

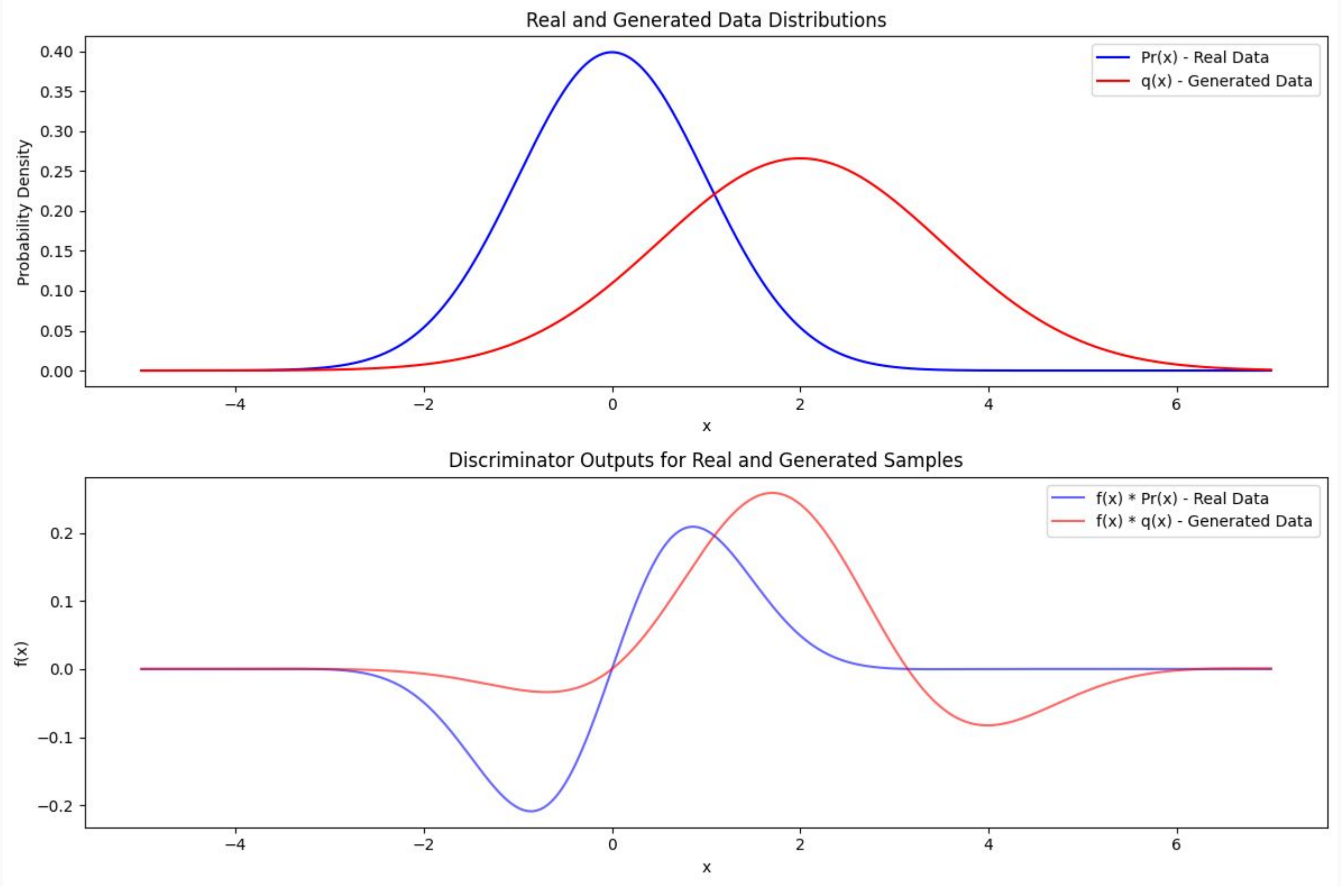
$$\begin{aligned} L[\boldsymbol{\phi}] &= \sum_j \mathrm{f}[\mathbf{x}^*_j, \boldsymbol{\phi}] - \sum_i \mathrm{f}[\mathbf{x}_i, \boldsymbol{\phi}] \\ &= \sum_j \mathrm{f}[\mathbf{g}[\mathbf{z}_j, \boldsymbol{\theta}], \boldsymbol{\phi}] - \sum_i \mathrm{f}[\mathbf{x}_i, \boldsymbol{\phi}], \end{aligned}$$

where we must constrain the neural network discriminator f[x_i,φ] to have an absolute gradient norm of less than one at every position x:

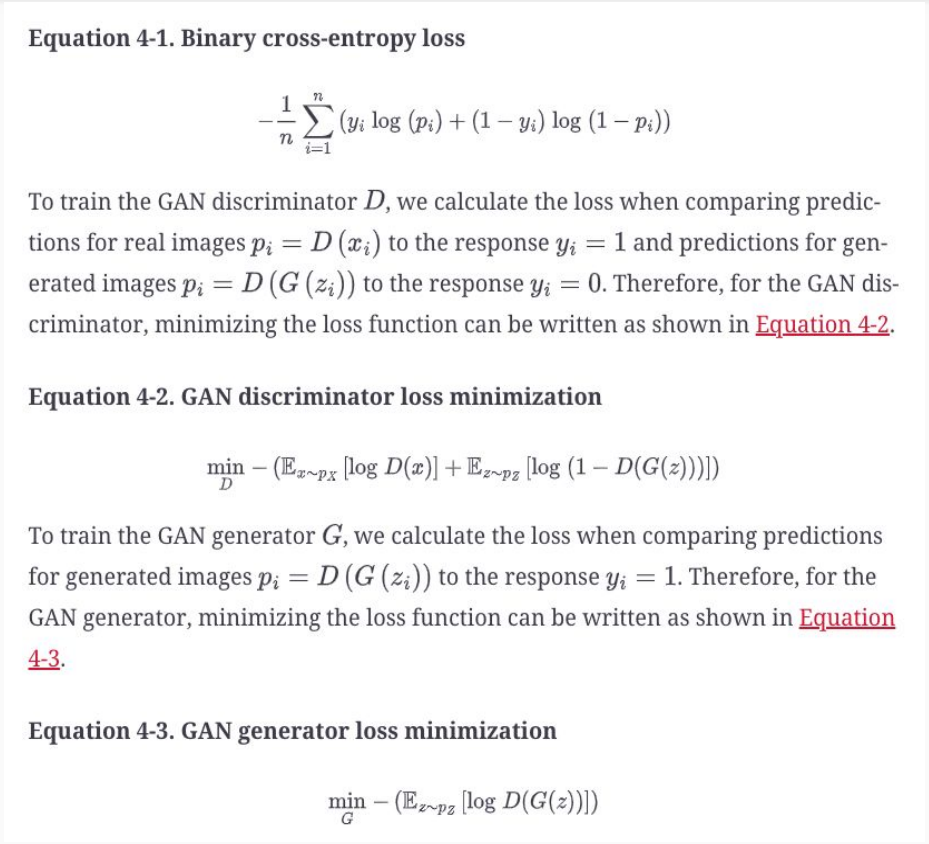
$$\left|\frac{\partial \mathrm{f}[\mathbf{x}, \boldsymbol{\phi}]}{\partial \mathbf{x}}\right| < 1.$$

*One way to achieve this is to clip the discriminator weights to a small range (e.g., [−0.01, 0.01]). An alternative is the gradient penalty WGAN-GP, which adds a regularization term that increases as the gradient norm deviates from unity.

Wasserstein Distance - Visualization



Discriminator and Generator



Equation 4-1. Binary cross-entropy loss

$$-\frac{1}{n}\sum_{i=1}^{n}\left(y_i \log\left(p_i\right) + \left(1-y_i\right)\log\left(1-p_i\right)\right)$$

To train the GAN discriminator D, we calculate the loss when comparing predictions for real images $p_i = D\left(x_i\right)$ to the response $y_i = 1$ and predictions for generated images $p_i = D\left(G\left(z_i\right)\right)$ to the response $y_i = 0$. Therefore, for the GAN discriminator, minimizing the loss function can be written as shown in Equation 4-2.

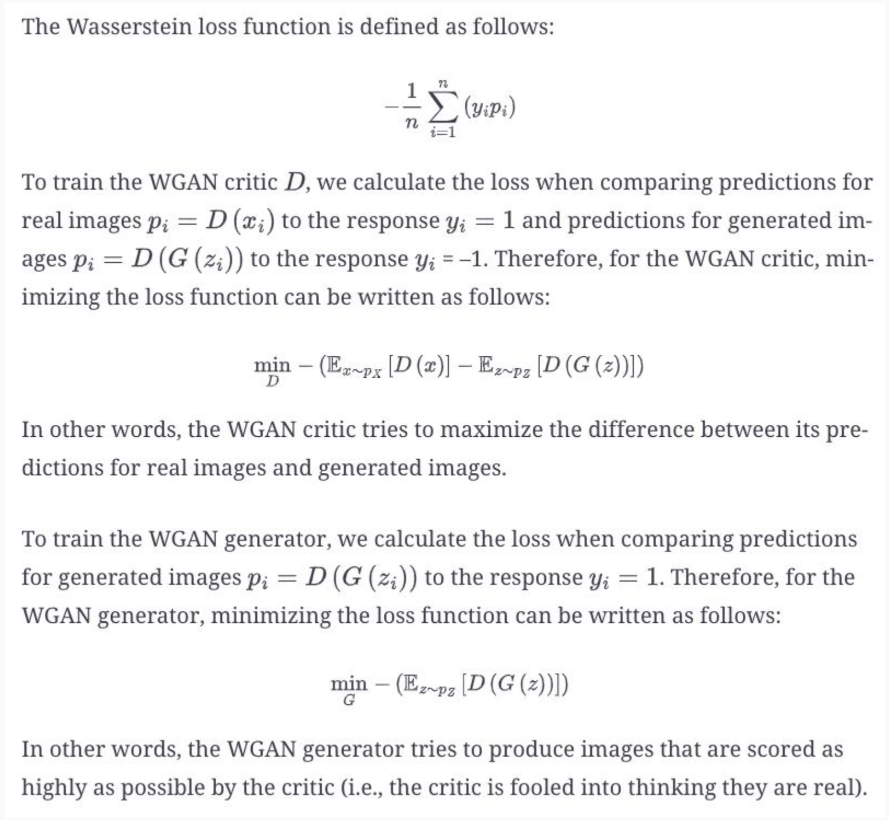
Equation 4-2. GAN discriminator loss minimization

$$\min_D -\left(\mathbb{E}_{x\sim p_X}\left[\log D(x)\right] + \mathbb{E}_{z\sim p_Z}\left[\log\left(1 - D(G(z))\right)\right]\right)$$

To train the GAN generator G, we calculate the loss when comparing predictions for generated images $p_i = D\left(G\left(z_i\right)\right)$ to the response $y_i = 1$. Therefore, for the GAN generator, minimizing the loss function can be written as shown in Equation 4-3.

Equation 4-3. GAN generator loss minimization

$$\min_G -\left(\mathbb{E}_{z\sim p_Z}\left[\log D(G(z))\right]\right)$$



The Wasserstein loss function is defined as follows:

$$-\frac{1}{n}\sum_{i=1}^{n}\left(y_i p_i\right)$$

To train the WGAN critic D, we calculate the loss when comparing predictions for real images $p_i = D\left(x_i\right)$ to the response $y_i = 1$ and predictions for generated images $p_i = D\left(G\left(z_i\right)\right)$ to the response y_i = –1. Therefore, for the WGAN critic, minimizing the loss function can be written as follows:

$$\min_D -\left(\mathbb{E}_{x\sim p_X}\left[D\left(x\right)\right] - \mathbb{E}_{z\sim p_Z}\left[D\left(G\left(z\right)\right)\right]\right)$$

In other words, the WGAN critic tries to maximize the difference between its predictions for real images and generated images.

To train the WGAN generator, we calculate the loss when comparing predictions for generated images $p_i = D\left(G\left(z_i\right)\right)$ to the response $y_i = 1$. Therefore, for the WGAN generator, minimizing the loss function can be written as follows:

$$\min_G -\left(\mathbb{E}_{z\sim p_Z}\left[D\left(G\left(z\right)\right)\right]\right)$$

In other words, the WGAN generator tries to produce images that are scored as highly as possible by the critic (i.e., the critic is fooled into thinking they are real).

Lipschitz constraint

A Lipschitz constraint is a mathematical property that limits how quickly a function can change with respect to changes in its input. In the context of GANs (Generative Adversarial Networks), the Lipschitz constraint is applied to the critic function (also called the discriminator) to ensure stable training and prevent certain issues like gradient vanishing or exploding.

A function f is said to be Lipschitz continuous if there exists a constant $K \geq 0$ such that for all x_1 and x_2 in the domain of f:

$$|f(x_1) - f(x_2)| \leq K \cdot |x_1 - x_2|$$

The smallest such K is called the Lipschitz constant of f. In other words, the Lipschitz constraint bounds the rate of change of the function's output with respect to changes in its input.

Lipschitz constraint - WGAN

- Wasserstein GAN allows the critic to output any real number, rather than restricting it to [0,1] with a sigmoid
- The key reason is that the Wasserstein loss cares about the relative scores the critic assigns to real vs fake images, rather than the absolute probabilities. As long as the critic consistently assigns a higher score to real images than fakes, the generator can use this to improve. The absolute scale of the scores doesn't matter.
- Allowing unbounded critic scores makes it more expressive - it doesn't have to squash all outputs into a small range. This flexibility lets it learn more complex features to distinguish real from fake.
- Critic scores could grow arbitrarily large. To prevent this, the Lipschitz constraint is introduced:

$$\frac{|D(x_1) - D(x_2)|}{|x_1 - x_2|} \leq 1$$

Numerator - the average pixel wise absolute difference between two images

Denominator - the absolute difference between the critic predictions

- This requires that the rate of change of the critic function is at most 1. Intuitively, the critic predictions cannot vary too rapidly from one image to a similar image. This prevents uncontrolled growth of the scores.

Enforcing the Lipschitz constraint

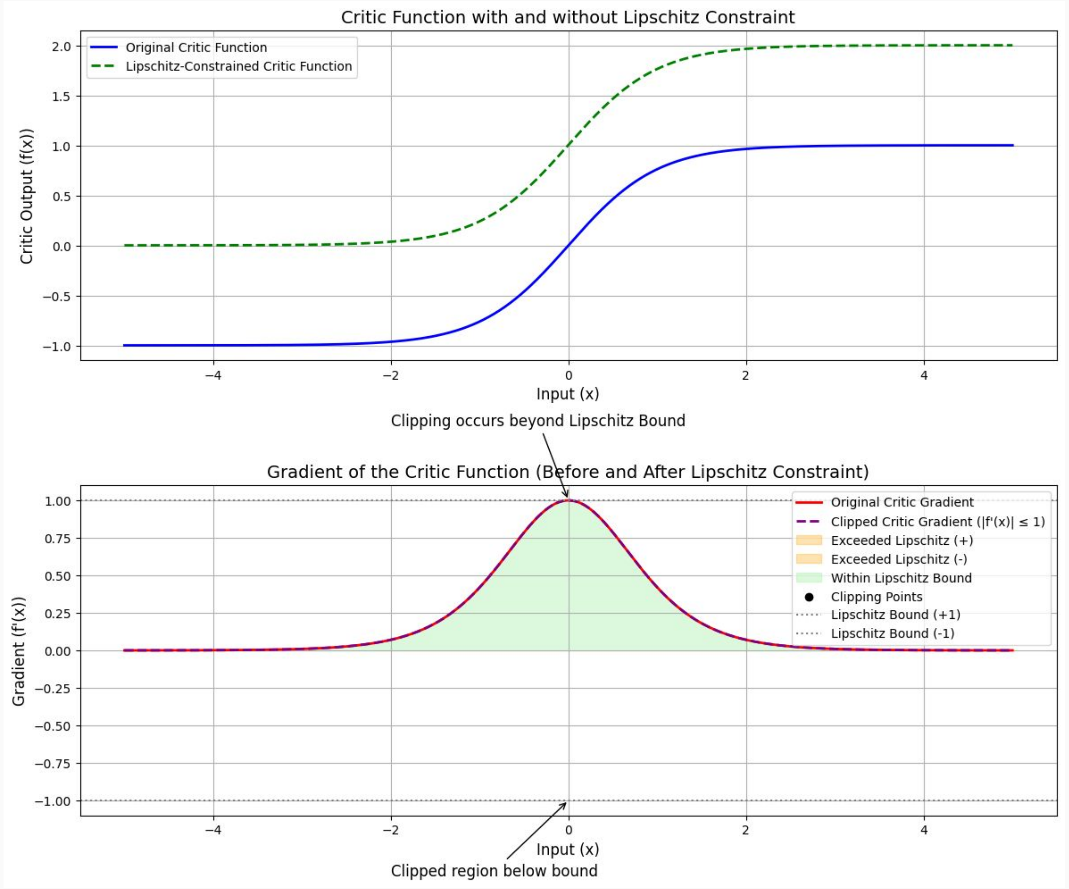
- In the context of WGANs (Wasserstein GANs), the critic function is required to be 1-Lipschitz, meaning that the Lipschitz constant K should be less than or equal to 1.
- This constraint helps to ensure that the critic provides meaningful gradients to the generator during training.

Enforcing the Lipschitz constraint can be done through various methods, such as:

1. Weight clipping: Limiting the weights of the critic's neural network to a specific range (e.g., [-c, c]).
2. Gradient penalty: Adding a term to the critic's loss function that penalizes the deviation of the critic's gradient norm from 1, as in the WGAN Gradient Penalty.

By enforcing the Lipschitz constraint, WGANs aim to achieve more stable training, improved sample quality, and reduced mode collapse compared to the original GAN formulation.

Visualizing the Lipschitz constraint



Wasserstein GAN

- Standard GAN training involves minimizing a divergence like Jensen-Shannon (JS) between the model distribution P_θ and real data distribution P_r .
- But these divergences can be discontinuous or fail when P_r and P_θ have disjoint supports, which happens when modeling low-dimensional distributions.
- The Earth Mover (EM) or Wasserstein-1 distance $W(P_r, P_\theta)$ matrizes a much weaker topology and is more sensible as an objective when learning manifold distributions.
- $W(P_r, P_\theta)$ is continuous and differentiable almost everywhere under mild assumptions

Mechanics

1. WGANs minimize an approximation of $W(P_r, P_\theta)$ by training a critic function f_w to maximize the difference between expectations over P_r and P_θ, subject to a Lipschitz constraint on f_w. This allows training f_w till optimality.
2. Training the critic till optimality provides more reliable gradients to the generator, avoiding vanishing gradients.
3. It also avoids mode collapse since the optimal generator cannot just focus on a few collapsed points that fool a non-optimal critic

Result

1. Empirically, the WGAN loss correlates better with sample quality during training compared to the saturated JS loss used by standard GANs. WGAN training is also more robust to architecture choices.
2. While weight clipping is used to enforce the Lipschitz constraint, better methods are needed.
3. But overall, WGANs provide a principled and practical approach to train generators modeling distributions supported on low-dimensional manifolds, with stability and consistency benefits.

Optimizations for WGANs

Issues with WGAN

- Wasserstein GAN (WGAN) makes progress toward stable training of GANs
- However, it sometimes can still generate only poor samples or fail to converge.
- These problems are often due to the use of weight clipping in WGAN to enforce a Lipschitz constraint on the critic, which can lead to undesired behavior
  - Such as capacity underuse - the clipped critic is biased towards simpler functions
  - And, can also cause exploding or vanishing gradients

WGAN Gradient Penalty

- An alternative way to enforce the Lipschitz constraint - by penalizing the gradient norm of the critic directly w.r.t. its input.
- Instead of clipping weights, they add a penalty term on how much the output of the critic can change in response to a change in input.

$$L = \underbrace{\mathop{\mathbb{E}}_{\tilde{\boldsymbol{x}} \sim \mathbb{P}_g} [D(\tilde{\boldsymbol{x}})] - \mathop{\mathbb{E}}_{\boldsymbol{x} \sim \mathbb{P}_r} [D(\boldsymbol{x})]}_{\text{Original critic loss}} + \underbrace{\lambda \mathop{\mathbb{E}}_{\hat{\boldsymbol{x}} \sim \mathbb{P}_{\hat{\boldsymbol{x}}}} \left[(\|\nabla_{\hat{\boldsymbol{x}}} D(\hat{\boldsymbol{x}})\|_2 - 1)^2 \right]}_{\text{Our gradient penalty}} .$$

Where:

L is the overall loss function for the critic,

D is the critic/discriminator,

$\tilde{x}$ is a fake/generated data sample from generator distribution P_g,

x is a real data sample from real data distribution P_r,

$\hat{x}$ is a random sample from distribution $\hat{P}_x$ along the interpolation lines between real and fake data samples,

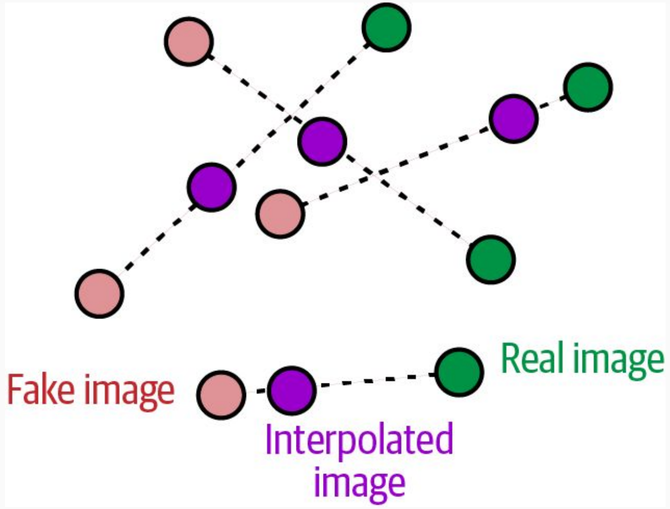
λ is the gradient penalty coefficient (a hyperparameter),

$\|\nabla_{\hat{x}} D(\hat{x})\|^2$ is the squared gradient norm of the critic's output with respect to the input $\hat{x}$.

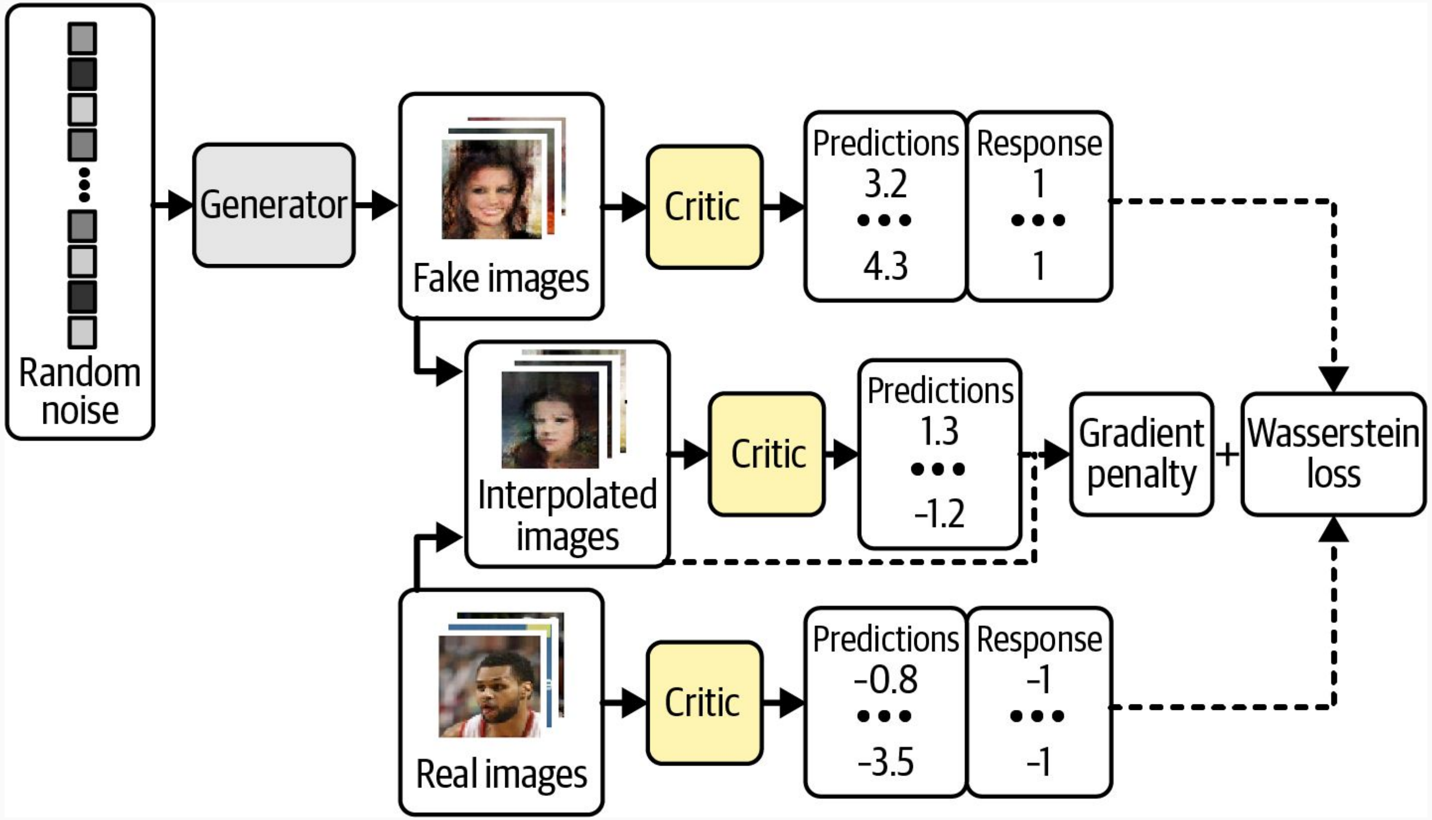
Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V., & Courville, A. C. (2017). Improved training of wasserstein gans. *Advances in neural information processing systems*, *30*.

Interpolated points

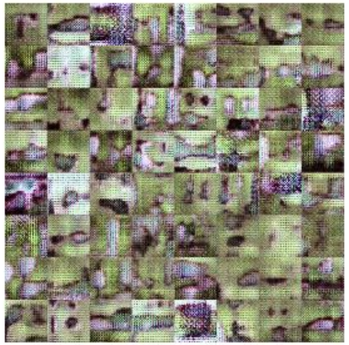
- The gradient penalty term measures the difference between the gradient norm for the critic's predictions with respect to the input images, and 1.
- By minimizing this term, the model is inclined to satisfy the Lipschitz constraint which requires that gradient norm to be ≤ 1 everywhere.
- It is computationally infeasible to calculate this gradient norm everywhere.
- Instead, the gradient penalty is estimated for some interpolated points between pairs of real and fake images.
- These interpolated points lie along the line connecting real and fake image pairs.

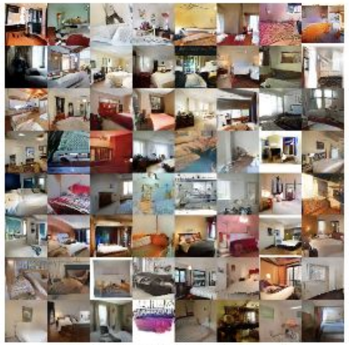


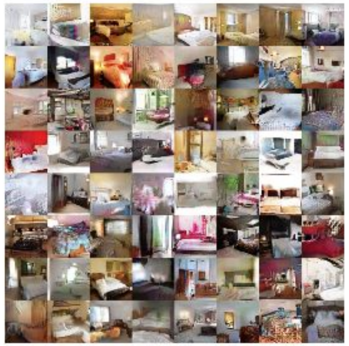
Gradient Penalty

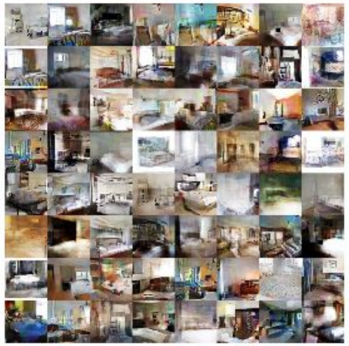


WGAN-GP Results


Method: WGAN with clipping
101-layer ResNet G and D


Method: WGAN-GP (ours)
G: DCGAN, D: DCGAN


Method: WGAN-GP (ours)
G: No BN and const. filter count


Method: WGAN-GP (ours)
G: 4-layer 512-dim ReLU MLP

Key Takeaways

- **Original GAN**: The critic can become overly confident, leading to unstable gradients that are not helpful for the generator.
- **WGAN with Weight Clipping**: Weight clipping tries to enforce the Lipschitz constraint, but it can lead to other issues, such as vanishing or exploding gradients.
- **WGAN with Gradient Penalty (WGAN-GP)**: Applying a gradient penalty directly addresses the issue by ensuring that the critic's gradient norm remains close to 1, providing more meaningful gradients and leading to more stable GAN training.

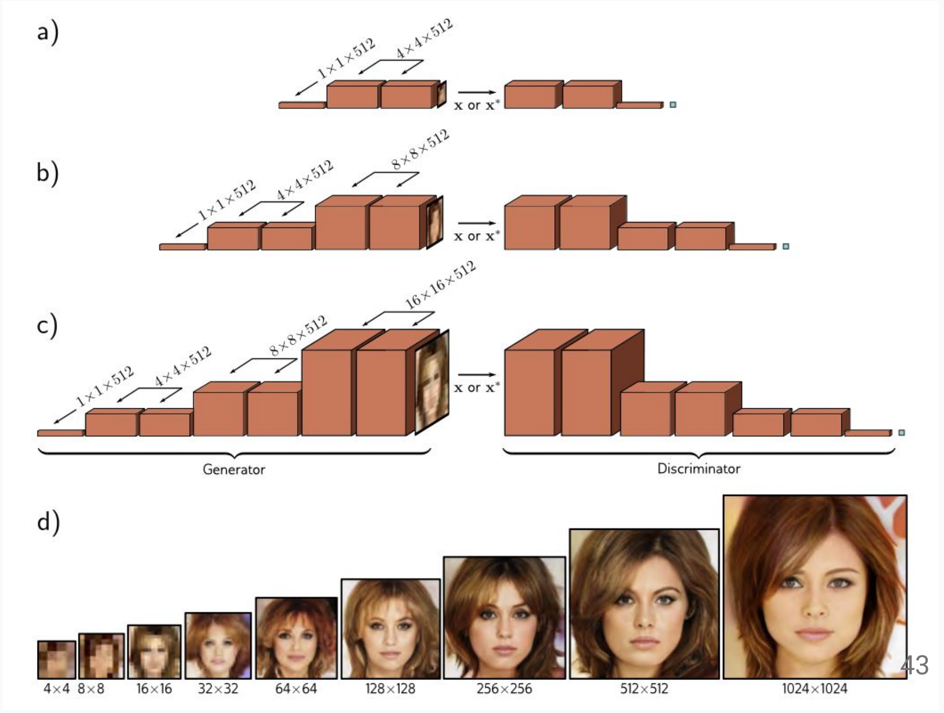
Progressive Growing

a) The generator is initially trained to create very small (4×4) images, and the discriminator to identify if these images are synthesized or downsampled real images.

b) After training at this low-resolution terminates, subsequent layers are added to the generator to generate (8×8) images. Similar layers are added to the discriminator to downsample back again.

c) This process continues to create (16×16) images and so on.

d) Images of increasing resolution generated at different stages from the same latent variable.



Adapted from Wolf (2021), using method of Karras et al. (2018).

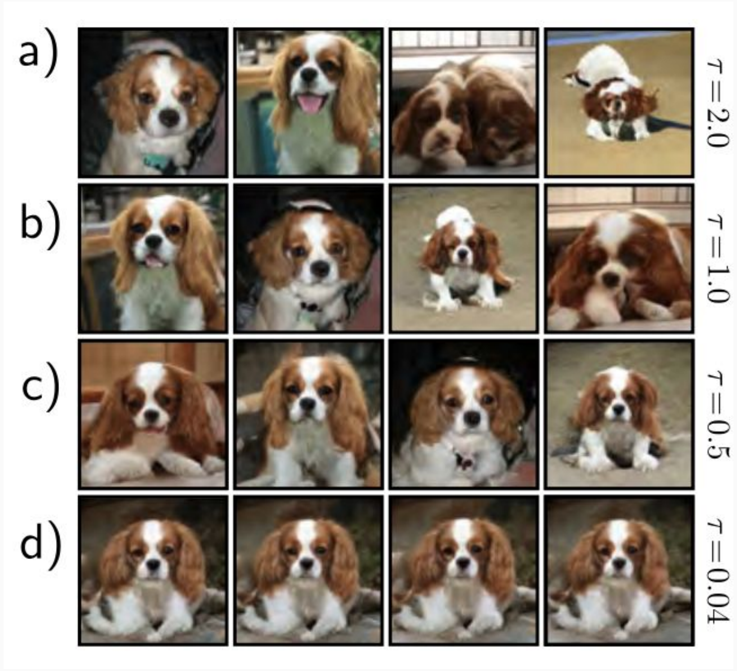
Truncation

The quality of GAN samples can be traded off against diversity by rejecting samples from the latent variable z that fall further than τ standard deviations from the mean.

a) If this threshold is large (τ = 2.0), the samples are visually varied but may have defects.

b–c) As this threshold is decreased, the average visual quality improves, but the diversity decreases.

d) With a very small threshold, the samples look almost identical. By judiciously choosing this threshold, it's possible to increase the average quality of GAN results.



Adapted from Brock et al. (2019)

Mini-batch discrimination

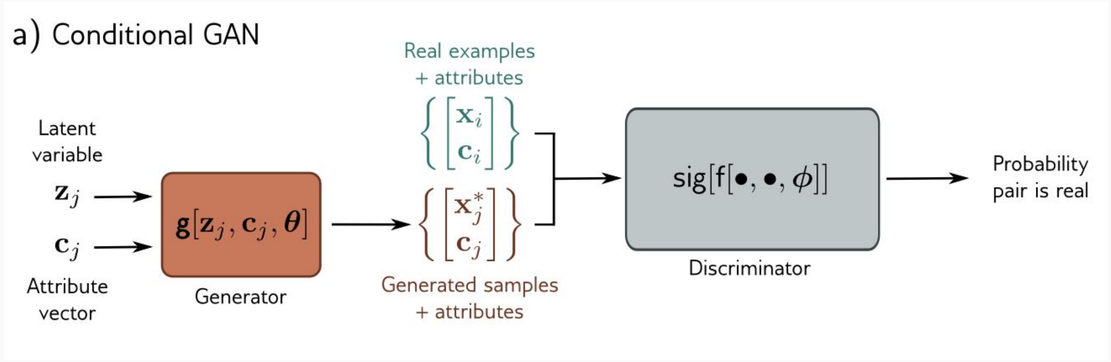
- Mini-batch discrimination ensures that the samples have sufficient variety and hence helps prevent mode collapse.
- This can be done by computing feature statistics across the mini-batches of synthesized and real data.
- These can be summarized and added as a feature map (usually toward the end of the discriminator).
- This allows the discriminator to send a signal back to the generator, encouraging it to include a similar amount of variation in the synthesized data as in the original dataset.

Conditional Generative Adversarial Network (CGAN)

Conditional GAN

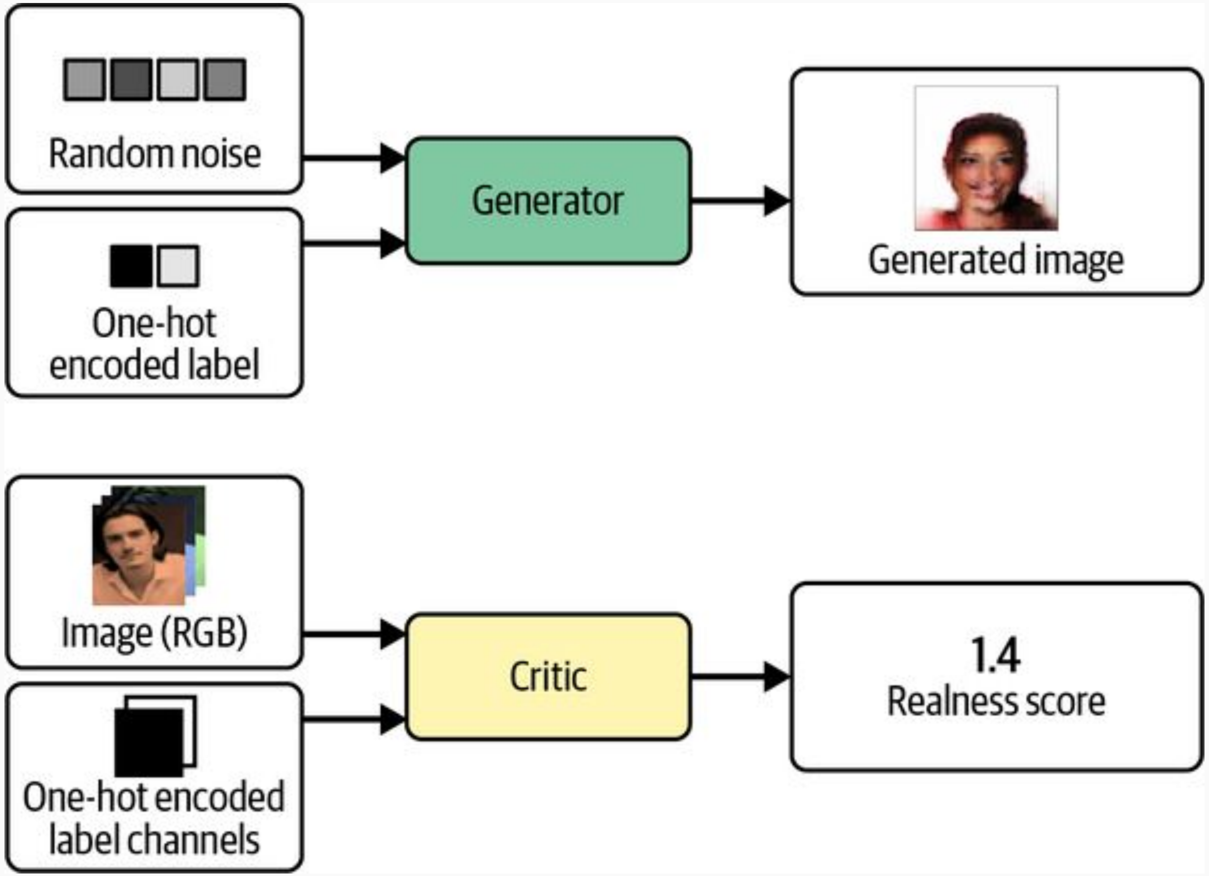
- Passes a vector c of attributes to both the generator and discriminator - g[z, c, θ] and f[x, c, φ], respectively.
- The generator aims to transform the latent variable z into a data sample x with the correct attribute c.
- The discriminator’s goal is to distinguish between the generated sample with the target attribute or a real example with the real attribute
- For the generator, the attribute c can be appended to the latent vector z.
- For the discriminator, it may be appended to the input if the data are 1D.
- If the data comprise images, the attribute can be linearly transformed to a 2D representation and appended as an extra channel to the discriminator input or to one of its intermediate hidden layers.

Conditional GAN


a) Conditional GAN
Real examples + attributes
$\left\{\begin{bmatrix}\mathbf{x}_i\\ \mathbf{c}_i\end{bmatrix}\right\}$
Latent variable
$\mathbf{z}_j$
$\mathbf{c}_j$
Attribute vector
$\mathbf{g}[\mathbf{z}_j, \mathbf{c}_j, \boldsymbol{\theta}]$
Generator
$\left\{\begin{bmatrix}\mathbf{x}^*_j\\ \mathbf{c}_j\end{bmatrix}\right\}$
Generated samples + attributes
sig[f[•, •, ϕ]]
Discriminator
Probability pair is real

- The generator of the conditional GAN also receives an attribute vector c describing some aspect of the image.
- As usual, the discriminator receives either a real example or a generated sample, but now it also receives the attribute vector;
- this encourages the samples both to be realistic and compatible with the attribute.

CGAN



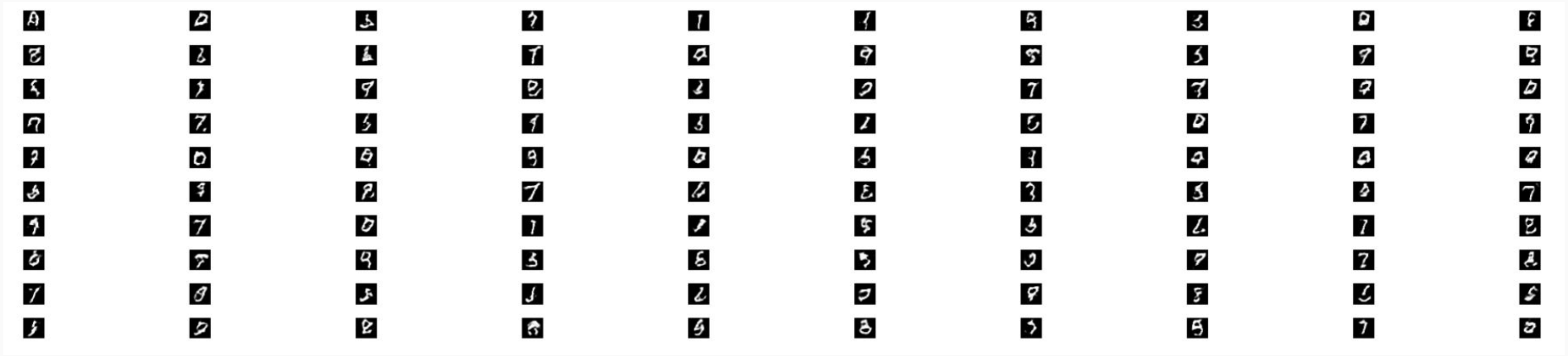
Random noise
One-hot encoded label
Generator
Generated image
Image (RGB)
One-hot encoded label channels
Critic
1.4
Realness score

CGAN with GP 50,000 Epochs - MINST Dataset

```
latent_dim = 10000
```

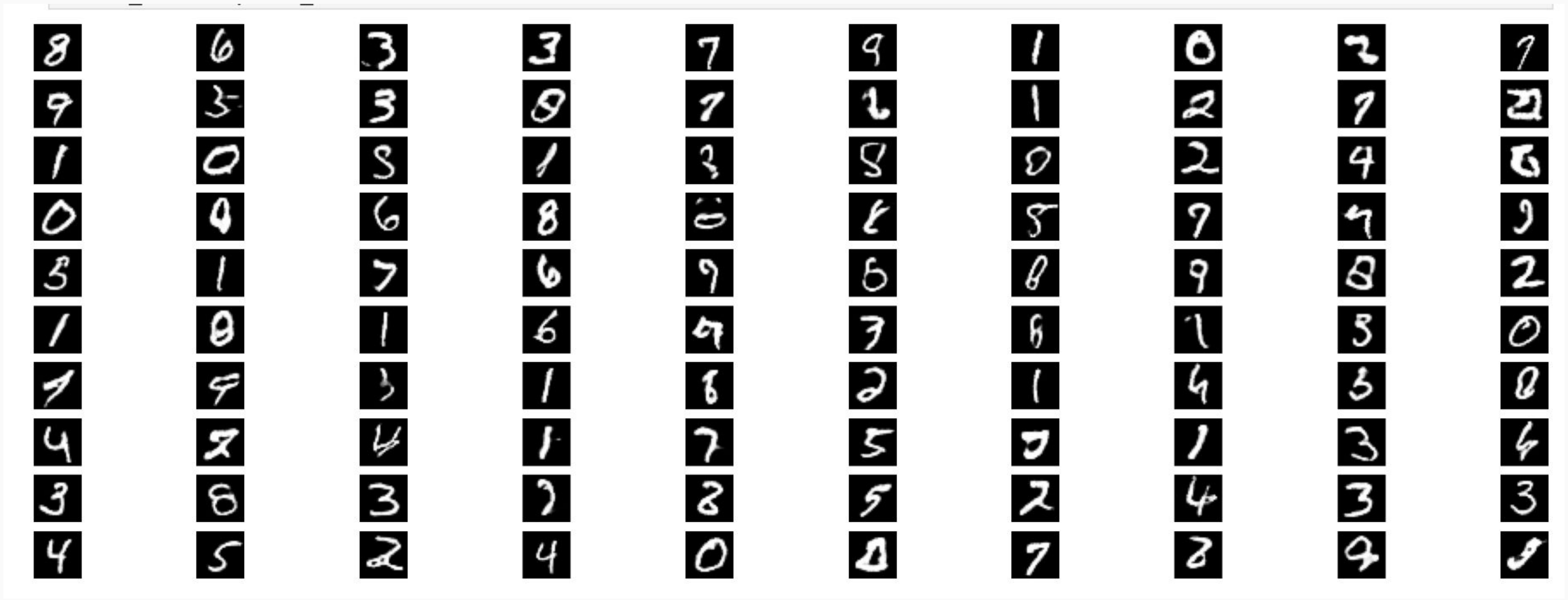
CGAN with GP 30 Epochs - MINST Dataset

```
latent_dim = 100
```



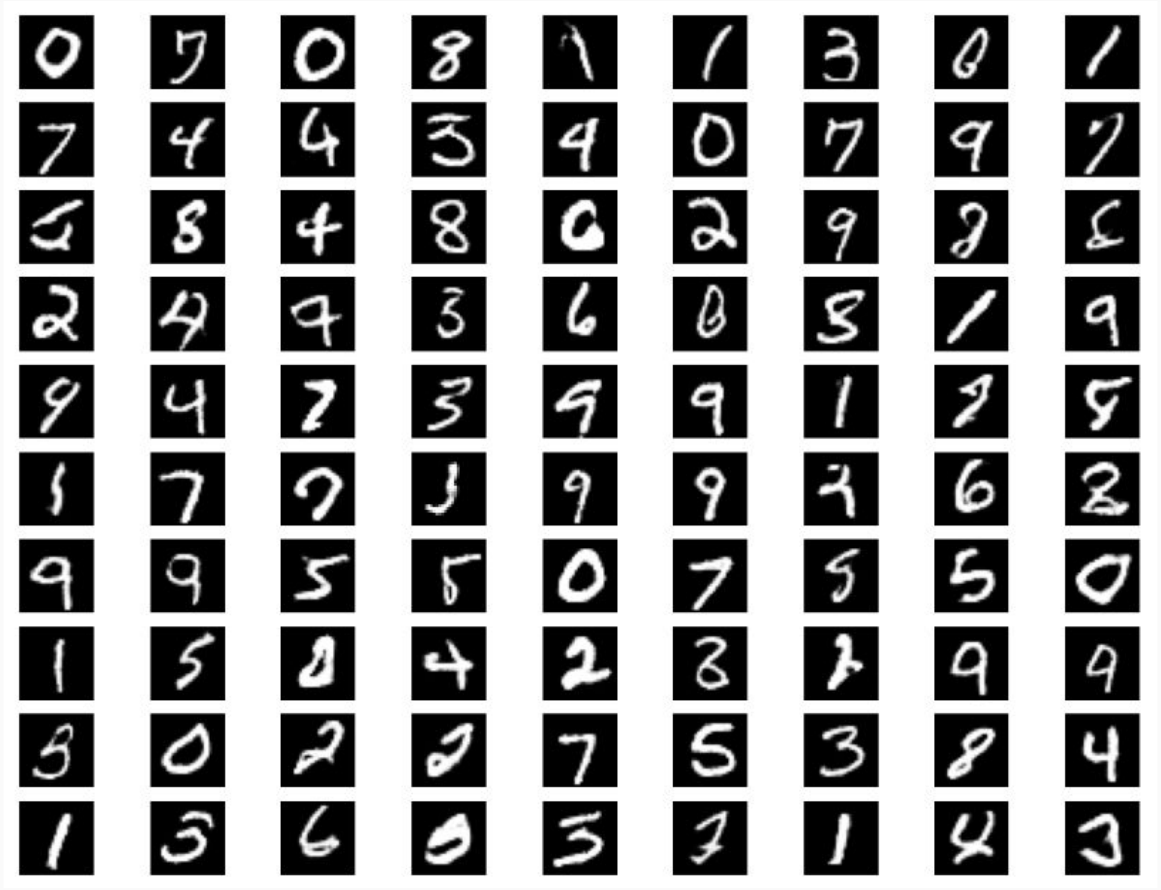
CGAN with GP 1,000 Epochs - MINST Dataset

```
latent_dim = 100
```

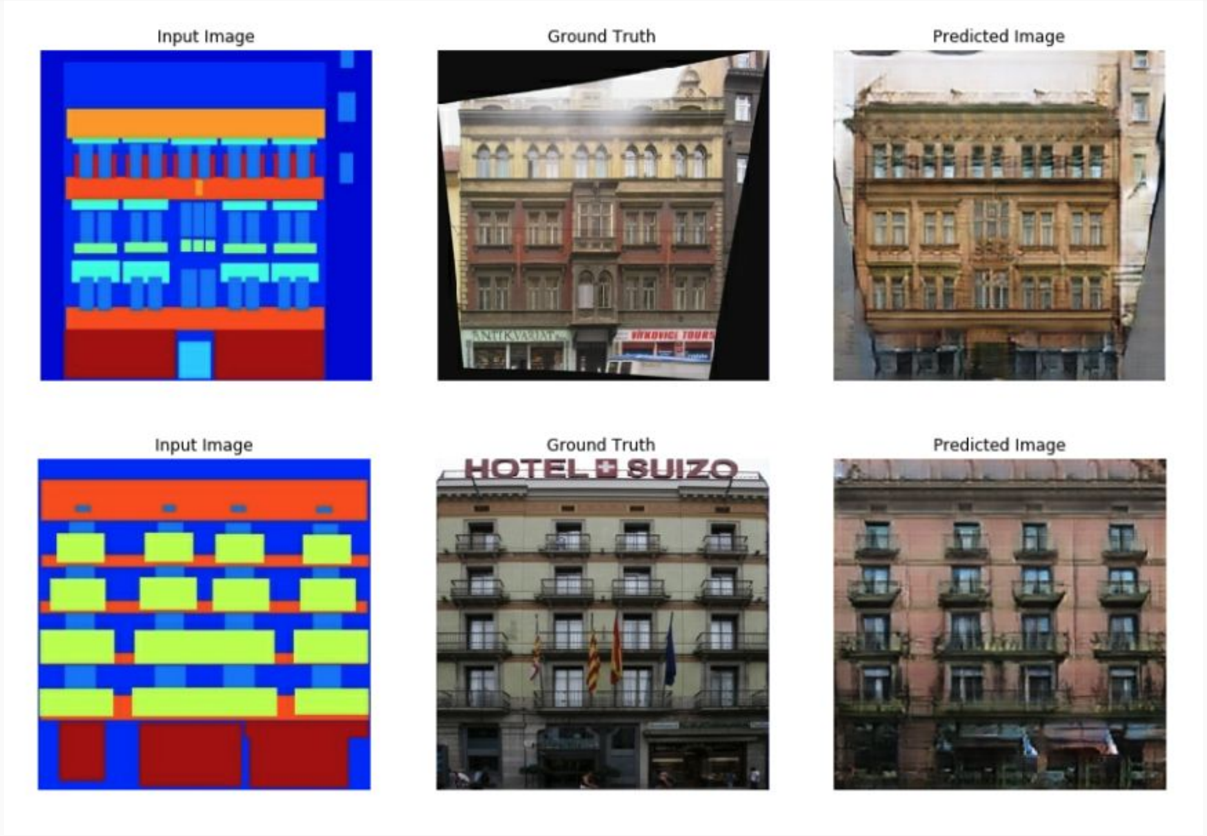


CGAN with GP 10,000 Epochs - MINST Dataset

```
latent_dim = 100
```



Pix2Pix Model - Image-to-image translation with a conditional GAN



https://www.tensorflow.org/tutorials/generative/pix2pix

Pix2Pix Model - Before and After Images for Training

Generator Architecture

- The generator is a convolutional autoregressive U-Net model.
- It takes as input an image c (e.g. grayscale image) and outputs an image x (e.g. colorized version).
- There are convolutional downsampling layers that capture context and upsampling layers that allow precise localization.
- Skip connections between the corresponding downsampling and upsampling layers provide local details to the global context.

Prince, S. J. (2023). *Understanding Deep Learning*. MIT press.

Objective Functions

- Content Loss: An L1 loss between input c and output x encourages them to match structurally. This stabilizes training.
- Adversarial Loss: A PatchGAN discriminator tries to classify if overlapping image patches in (c, x) are real or fake. This makes the local texture of x realistic.
- The PatchGAN discriminator operates convolutionally, classifying each N x N patch in the image. The outcomes are spatially averaged.
- Generator tries to minimize content loss (match structure to c) and fool the PatchGAN (make local patches in x look real).

Noise and Stochasticity

- Unlike regular GAN generators, no noise vector z is used as input.
- The input image c provides all the structure and guidance needed.
- When noise vectors were added, the generator learned to ignore them, since matching c provides a stronger training signal.
- Adding noise actually hurts output quality since x ignores the guidance in c.
- In summary,
    - the U-Net generator in Pix2Pix relies completely on the input image to guide the structure of the output.
    - The adversarial PatchGAN loss is only used to make the local texture details look real, not to introduce stochasticity.
    - This allows high quality image-to-image translation guided by the input image structure.

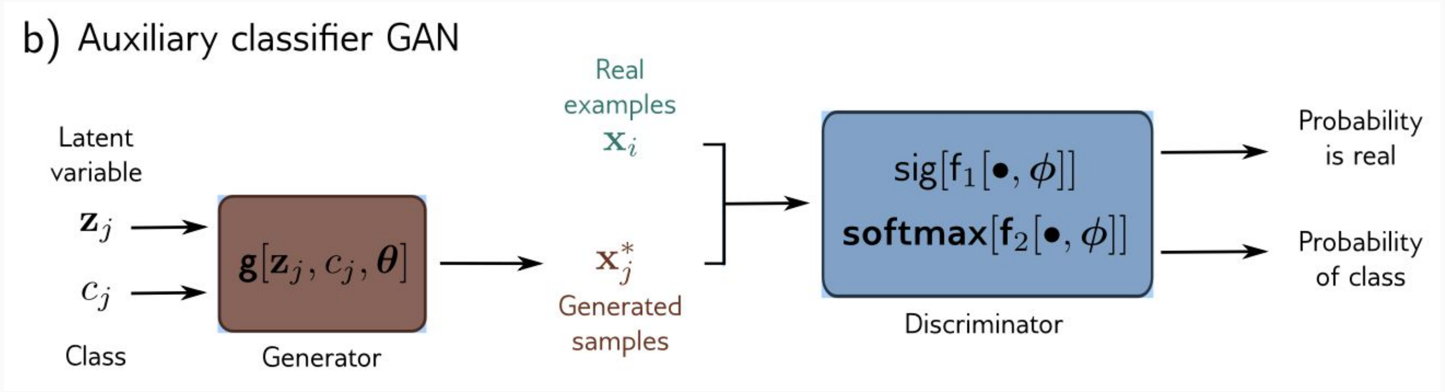
Prince, S. J. (2023). *Understanding Deep Learning*. MIT press.

Auxiliary classifier GAN (ACGAN)

Auxiliary classifier GAN (ACGAN)

- The auxiliary classifier GAN or ACGAN simplifies conditional generation by requiring that the discriminator correctly predicts the attribute .
- For a discrete attribute with C categories, the discriminator takes the real/synthesized image as input and has C + 1 outputs
- the first is passed through a sigmoid function and predicts if the sample is generated or real.
- The remaining outputs are passed through a softmax function to predict the probability that the data belongs to each of the C classes.
- Networks trained with this method can synthesize multiple classes from ImageNet

Auxiliary classifier GAN (ACGAN)


b) Auxiliary classifier GAN
Latent variable
$\mathbf{z}_j$
c_j
Class
$\mathbf{g}[\mathbf{z}_j, c_j, \boldsymbol{\theta}]$
Generator
Real examples $\mathbf{x}_i$
$\mathbf{x}_j^*$
Generated samples
$\text{sig}[\text{f}_1[\bullet, \phi]]$
$\textbf{softmax}[\textbf{f}_2[\bullet, \phi]]$
Discriminator
Probability is real
Probability of class

The generator of the auxiliary classifier GAN (ACGAN) takes a discrete attribute variable.

The discriminator must both

(i) determine if its input is real or synthetic and

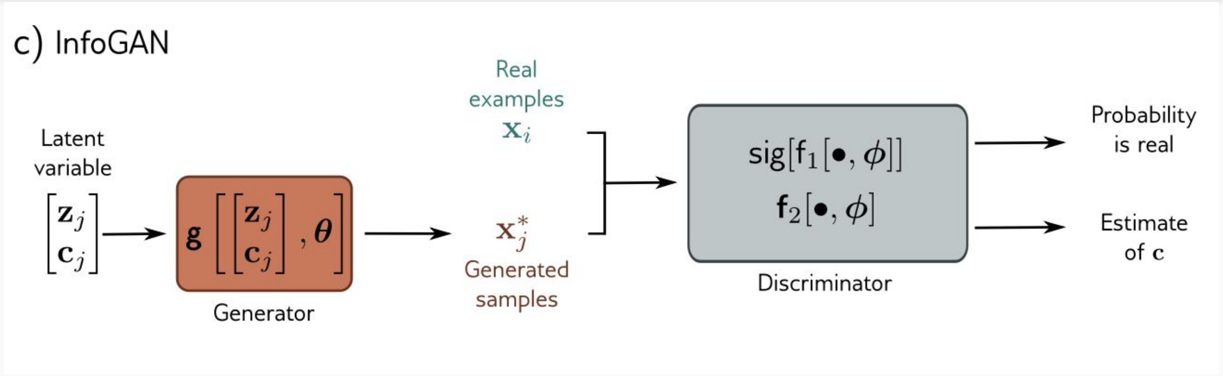
(ii) identify the class correctly.

InfoGAN

InfoGAN

- Attempts to identify important attributes automatically.
- The generator takes a vector consisting of random noise variables z and random attribute variables c.
- The discriminator both predicts whether the image is real or synthesized and estimates the attribute variables.
- The insight is that interpretable real-world characteristics should be easiest to predict and hence will be represented in the attribute variables c.
- The attributes in c may be discrete (and a binary or multiclass cross-entropy loss would be used) or continuous (and a least squares loss would be used).
- The discrete variables identify categories in the data, and the continuous ones identify gradual modes of variation.

InfoGAN


c) InfoGAN
Latent variable
$\begin{bmatrix} \mathbf{z}_j \\ \mathbf{c}_j \end{bmatrix}$
$\mathbf{g}\left[\begin{bmatrix} \mathbf{z}_j \\ \mathbf{c}_j \end{bmatrix}, \boldsymbol{\theta}\right]$
Generator
Real examples $\mathbf{x}_i$
$\mathbf{x}^*_j$ Generated samples
$\text{sig}[\mathrm{f}_1[\bullet, \phi]]$
$\mathbf{f}_2[\bullet, \phi]$
Discriminator
Probability is real
Estimate of $\mathbf{c}$

- The InfoGAN splits the latent variable into noise z and unspecified random attributes c.
- The discriminator must distinguish if its input is real and also reconstruct these attributes.
- In practice, this means that the variables c correspond to salient aspects of the data with real-world interpretations (i.e., the latent space is disentangled).

Prince, S. J. (2023). *Understanding Deep Learning*. MIT press.

CycleGAN

CycleGAN - data with distinct styles but no matching pairs.

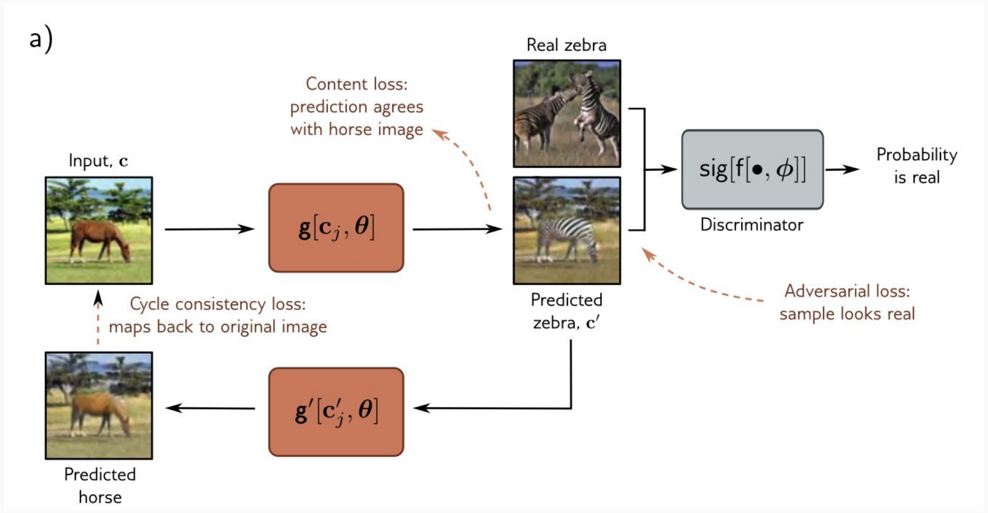
The CycleGAN is designed to learn image-to-image translation between two domains without any paired training examples.

1. Two generator networks are used: $G: X \rightarrow Y$ and $F: Y \rightarrow X$. These generate images in the opposite domains.
2. An adversarial loss is applied to each generator using two discriminator networks D_X and D_Y. This loss encourages the generated images $G(x)$ and $F(y)$ to be indistinguishable from real images y and x in the target domains.
3. A cycle consistency loss enforces reversibility. For each image x from domain X, the image translation should be reversible: $x \rightarrow G(x) \rightarrow F(G(x)) \approx x$. The same for each image y from domain Y. This loop enforces realistic outputs and one-to-one mapping.
4. There is also a content loss between input and output images using L1 distance. This encourages perceptual similarity between input and output.
5. The full loss function is a weighted sum of the adversarial losses (enforcing realism), cycle-consistency losses (enforcing reversibility) and content losses (preserving perceptual similarity).
6. By combining these different objectives, the generators are trained to translate images across domains in both directions, while ensuring perceptual quality, reality of the outputs, and consistency between forward and backward mappings.

Prince, S. J. (2023). *Understanding Deep Learning*. MIT press.

CycleGAN - Mechanics

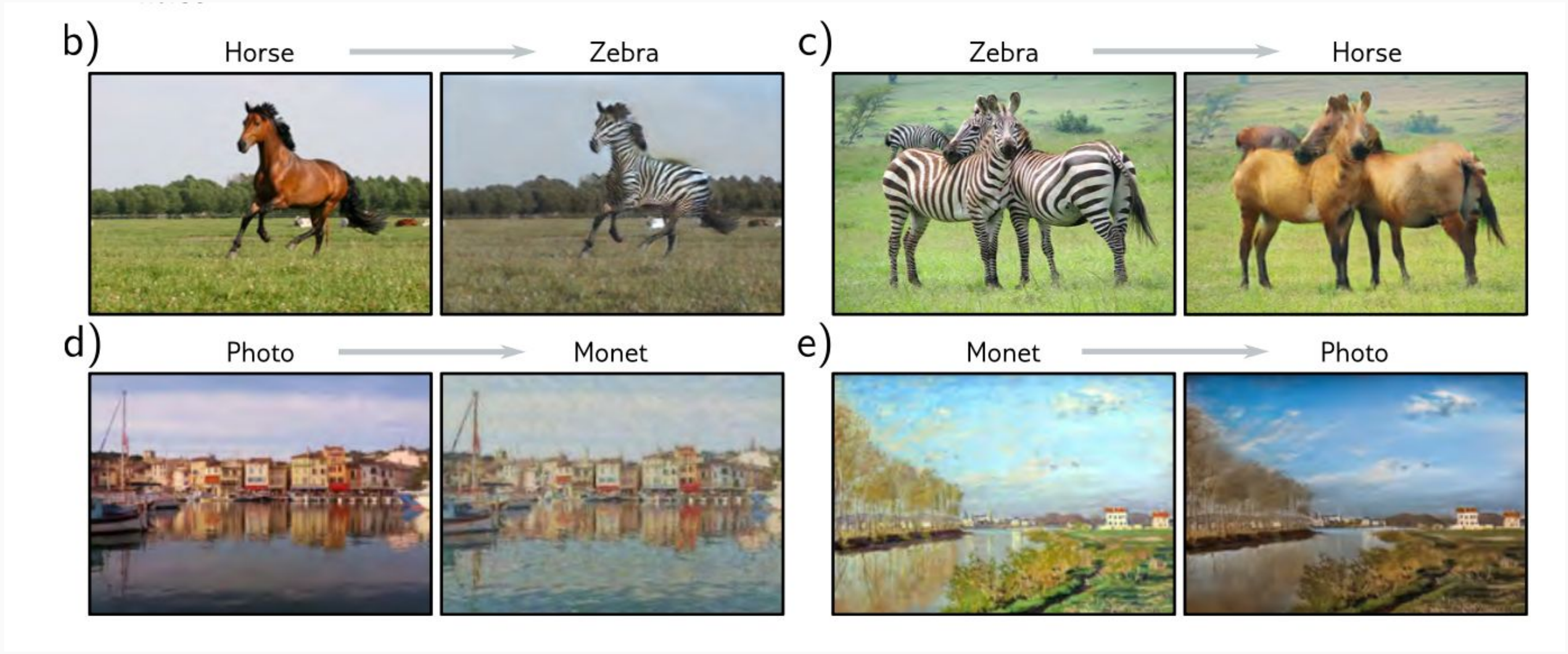
- The first c' = g[c_j,θ] translates from an image c in the first style (horse) to an image c' in the second style (zebra).
- The second model c = g'[c',θ] learns the opposite map- ping.
- The cycle consistency loss penalizes both models if they cannot successfully convert an image to the other domain and back to the original.
- Two adversarial losses encourage the translated images to look like realistic examples of the target domain (shown here for zebra only).
- Two content losses encourage the details and layout of the images before and after each mapping to be similar (i.e., the zebra is in the same position and pose that the horse was and against the same background and vice versa).



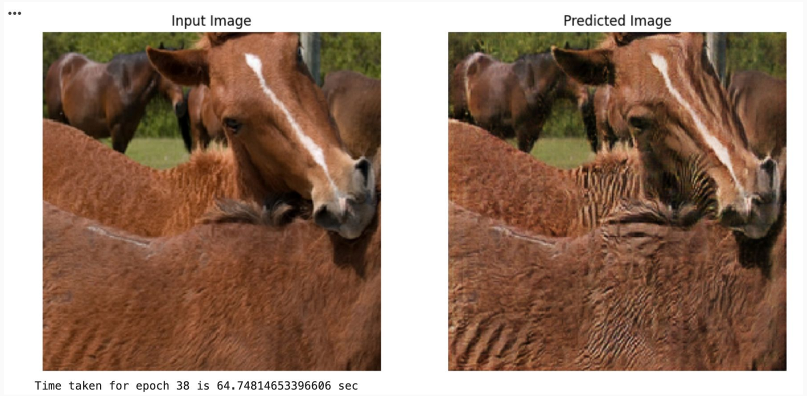
a)
Real zebra
Content loss: prediction agrees with horse image
Input, $\mathbf{c}$
$\mathbf{g}[\mathbf{c}_j, \boldsymbol{\theta}]$
$\text{sig}[f[\bullet, \phi]]$
Probability is real
Discriminator
Cycle consistency loss: maps back to original image
Predicted zebra, $\mathbf{c}'$
Adversarial loss: sample looks real
$\mathbf{g}'[\mathbf{c}'_j, \boldsymbol{\theta}]$
Predicted horse

Adapted from Zhu et al. (2017).

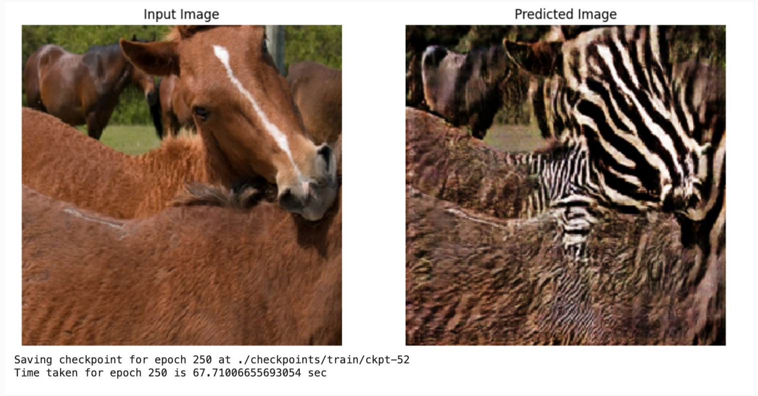
CycleGAN - Results



CycleGAN - Process



```
Time taken for epoch 38 is 64.74814653396606 sec
```



```
Saving checkpoint for epoch 250 at ./checkpoints/train/ckpt-52
Time taken for epoch 250 is 67.71006655693054 sec
```

StyleGAN

StyleGan - partitions the variation in a dataset into meaningful components, each of which is controlled by a subset of the latent variables.

StyleGAN introduces two sets of latent variables that are injected at different layers to independently control attributes at different scales:

- Noise vectors (z_1, z_2, etc.):
  - These are drawn from a normal distribution and control fine-grained details.
  - They are injected additively after each convolution layer.
  - Per-channel scaling factors ψ_i allow them to affect different feature maps differently.
  - As the spatial resolution increases in deeper layers, these noise vectors begin to control finer-scale details.
- Style vectors (y_1, y_2, etc.):
  - These are derived from a top-level style vector w, which goes through a mapping network to decorrelate its dimensions.
  - This style vector is transformed into y vectors that control feature map channel statistics via adaptive instance normalization (AdaIN).
  - The same w controls styles at all scales, while the y_i transform controls scale-specific variation.

Noise and Style

This structured injection of noise and style has several advantages:

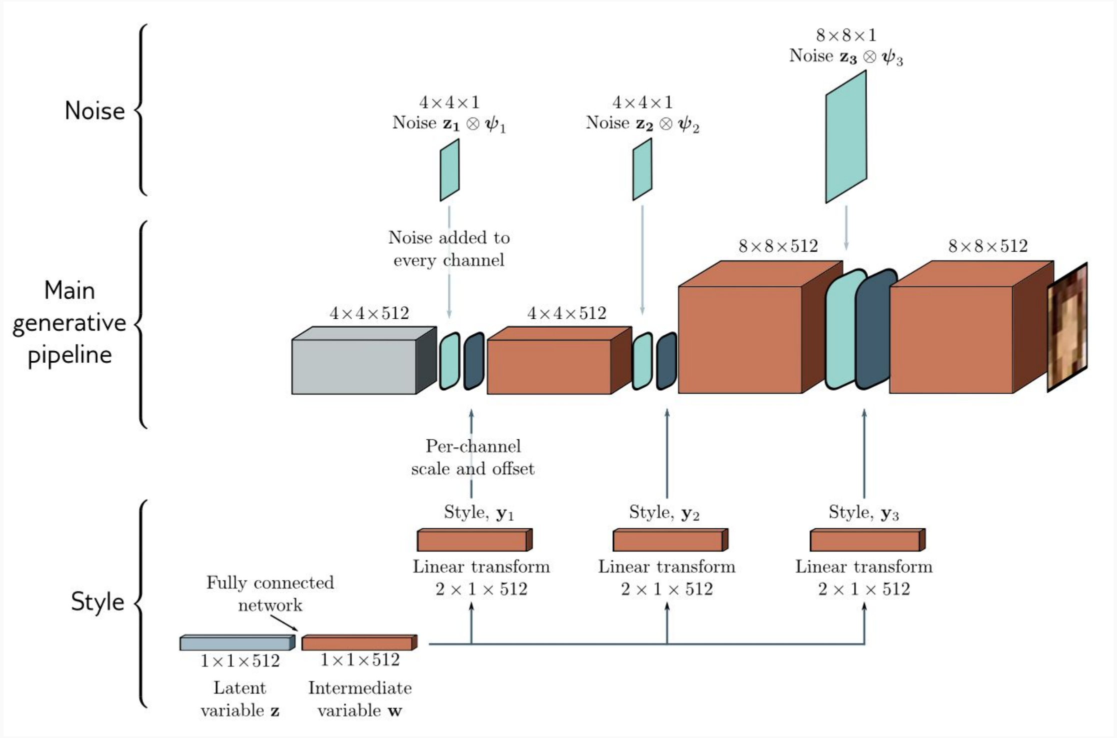
1. It disentangles style (y) from stochastic detail (z). Attribute changes can be made by modifying style dims without affecting image details.
2. Style reinjection at each scale leads to global consistency, while the noise vectors control local stochasticity at different resolutions.
3. Mapping network decorrelation allows different dims of w to independently control salient visual attributes like pose, hair style, etc.

In summary, StyleGAN carefully injects style and noise vectors across scales of the generator.

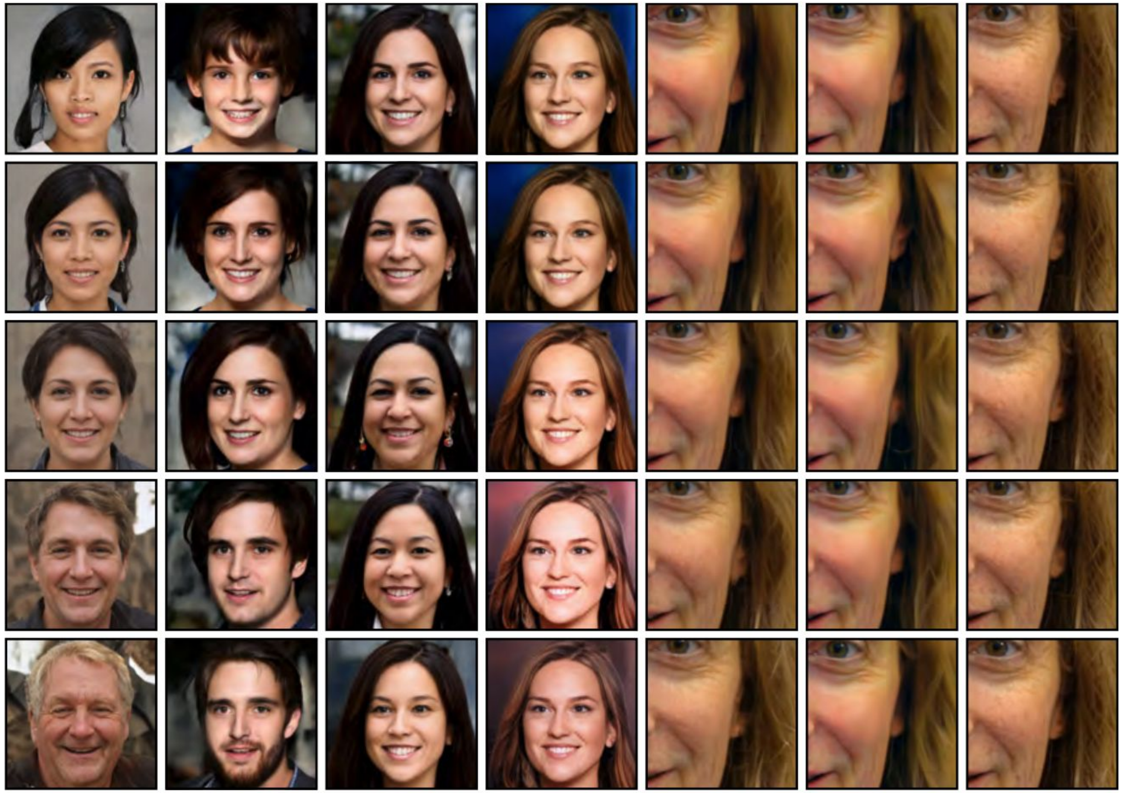
- This disentangles factors of variation, allowing fine-grained control over Stochastic variation (z) vs high-level attributes (w) that get applied globally via style reinjection (y).
- The result is an extremely expressive generator producing high-quality, diverse images.

StyleGAN

- The main pipeline (center row) starts with a constant learned representation (gray box).
- This is passed through a series of convolutional layers and gradually upsampled to create the output.
- Noise (top row) is added at different scales by periodically adding Gaussian variables z· with per-channel scaling ψ·.
- The Gaussian style variable z is passed through a fully connected network to create intermediate variable w (bottom row).
- This is used to set the mean and variance of each channel at various points in the pipeline.


Noise
Main generative pipeline
Style
$4\times4\times1$ Noise $\mathbf{z_1}\otimes\psi_1$
$4\times4\times1$ Noise $\mathbf{z_2}\otimes\psi_2$
$8\times8\times1$ Noise $\mathbf{z_3}\otimes\psi_3$
Noise added to every channel
$4\times4\times512$
$4\times4\times512$
$8\times8\times512$
$8\times8\times512$
Per-channel scale and offset
Style, $\mathbf{y}_1$
Style, $\mathbf{y}_2$
Style, $\mathbf{y}_3$
Linear transform $2\times1\times512$
Linear transform $2\times1\times512$
Linear transform $2\times1\times512$
Fully connected network
$1\times1\times512$ Latent variable $\mathbf{z}$
$1\times1\times512$ Intermediate variable $\mathbf{w}$

StyleGAN



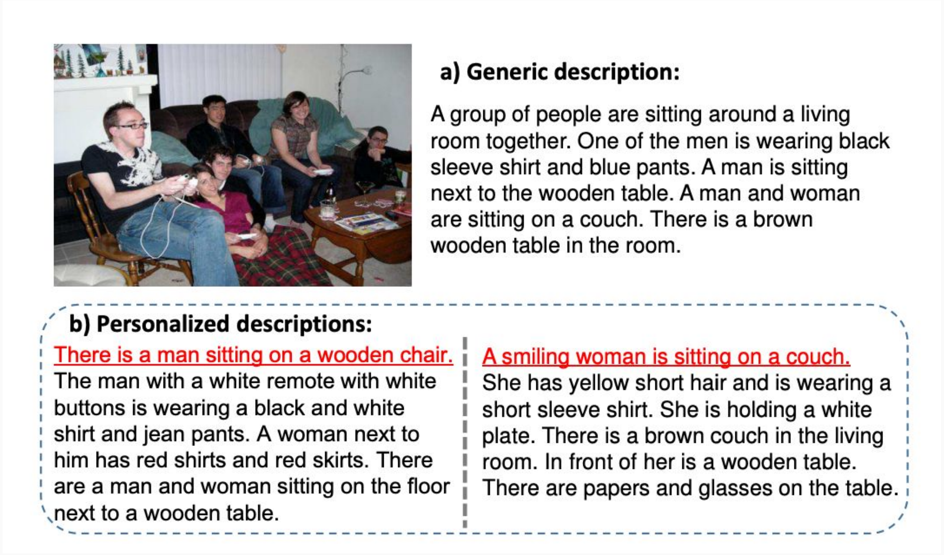
- First four columns show systematic changes in style at various scales.
- Fifth column shows the effect of increasing noise magnitude.
- Last two columns show different noise vectors at two different scales.

GAN for Visual Paragraph Generation

Generating coherent paragraphs of images and videos

Challenges:

- Existing methods are restricted by small biased annotated datasets.
- Lack of sufficient annotated image-paragraph pairs limits model generalization.
- Capturing detailed semantics and ensuring paragraph coherence is difficult.



a) Generic description:

A group of people are sitting around a living room together. One of the men is wearing black sleeve shirt and blue pants. A man is sitting next to the wooden table. A man and woman are sitting on a couch. There is a brown wooden table in the room.

b) Personalized descriptions:

There is a man sitting on a wooden chair. The man with a white remote with white buttons is wearing a black and white shirt and jean pants. A woman next to him has red shirts and red skirts. There are a man and woman sitting on the floor next to a wooden table.

A smiling woman is sitting on a couch. She has yellow short hair and is wearing a short sleeve shirt. She is holding a white plate. There is a brown couch in the living room. In front of her is a wooden table. There are papers and glasses on the table.

Liang, X., et. al (2017). Recurrent topic-transition gan for visual paragraph generation. In *Proceedings of the IEEE international conference on computer vision* (pp. 3362-3371).

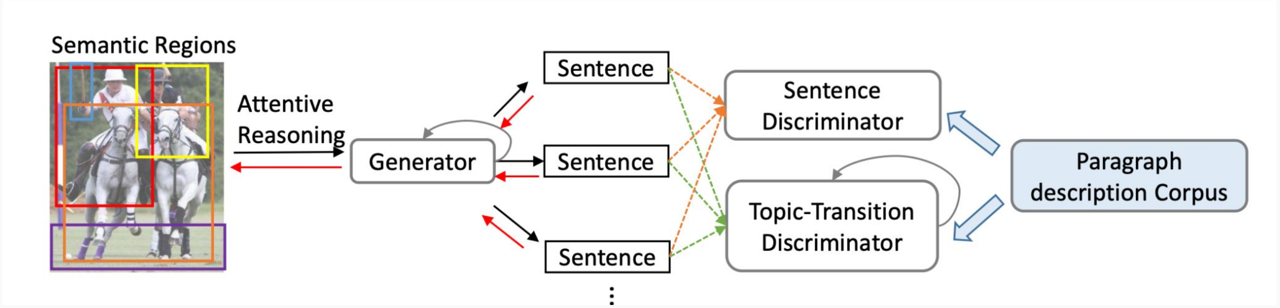
Training

- Uses an adversarial training scheme between a paragraph generator and multi-level discriminators to generate realistic and smooth paragraphs.
- Generator recurrently incorporates visual and language attention on local semantic regions to focus on details.
- Multi-level discriminators assess plausibility of individual sentences and topic coherence across sentences.
- Supports both supervised learning from annotated paragraphs and semi-supervised learning from just image captions.

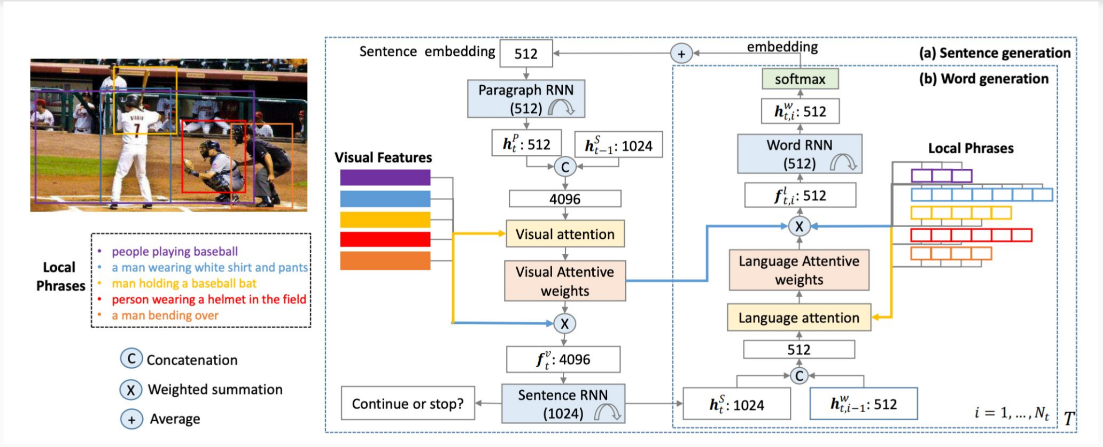
Liang, X., et. al (2017). Recurrent topic-transition gan for visual paragraph generation. In *Proceedings of the IEEE international conference on computer vision* (pp. 3362-3371).

Method Details

- Paragraph generator has a hierarchy of RNNs tracking paragraph, sentence and word states with attention models.
- Spatial visual attention selects relevant semantic regions based on current state.
- Language attention brings in related words from region captions.
- Sentence discriminator evaluates sentence plausibility.
- Topic-transition discriminator evaluates topic smoothness across sentences.
- Can generate personalized paragraphs by manipulating the first sentence.



Attention and Memory



1. It has a memory hP_t that keeps track of what has been generated so far in the paragraph. This memory gets updated after each sentence is created.
2. To generate the next sentence, it first looks at the image regions v_j and uses attention to pick the ones relevant for the next sentence f_t^v
3. This information goes into the sentence generator hS_t which decides if it should generate another sentence or stop.
4. To generate each word in the new sentence, it focuses on relevant words from the image regions using attention again f_t^l, i These words help it predict the next word w_t, i
5. After the sentence { $w_t, 1, ..., w_t, N_t$ } is complete, the overall paragraph memory hP_t is updated and the loop continues for the next sentence s_{t+1}

References

- Foster, D. (2022). *Generative deep learning*. " O'Reilly Media, Inc. - Chapter 4, 10.
- *Prince, S. J. (2023). Understanding Deep Learning. MIT press.* - Chapter 15

This week

1. Homework - Please run these two notebooks
    a. https://github.com/vijaygwu/SEAS8525/blob/main/Class_8_CGAN_MINST.ipynb
    b. https://github.com/vijaygwu/SEAS8525/blob/main/Class_8_WGAN_GP.ipynb
2. Finals in 2 weeks