Deep Learning for Computer Vision

Deep Generative Models in Vision: An Introduction

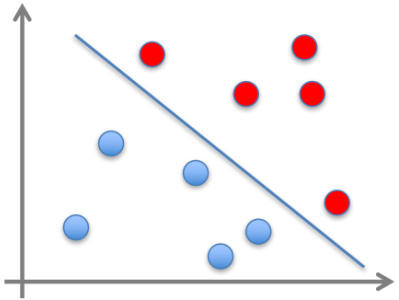
Vineeth N Balasubramanian

Department of Computer Science and Engineering
Indian Institute of Technology, Hyderabad

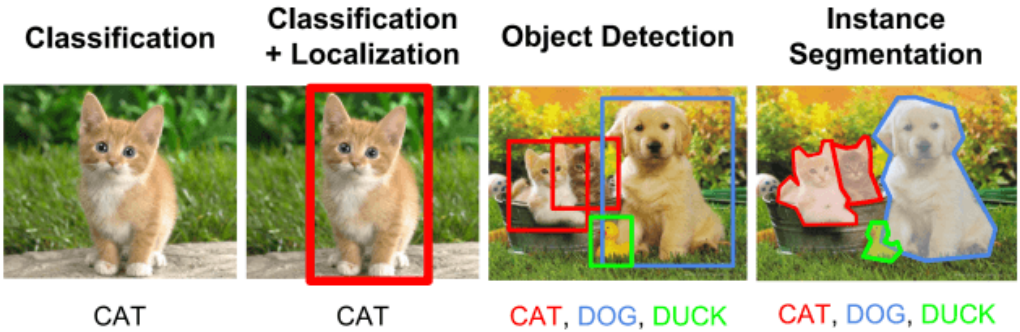
Supervised Learning

- Learning a mapping between data (inputs) to label (output).
- Data is given in the form of input-label pairs $\{(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)\}$.
- The goal is to learn a function to map x to y i.e., $f(x) = y$.

Let's see some examples!

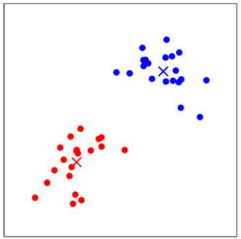


Supervised Learning: Examples in Computer Vision

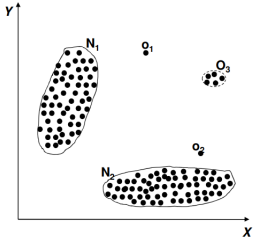


Credit: Fei Fei Li, J Johnson and S Yeung, CS231n, Stanford Univ

Going Beyond Supervised Learning



Unsupervised Learning/ Data Understanding

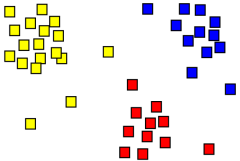


Detecting Outliers

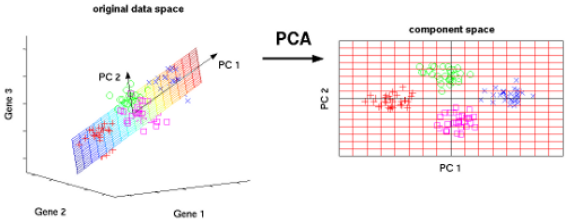
Generating Data

Unsupervised Learning

- Capturing the underlying structure of the data
- Only data (x) is provided; since no labels are required, training data is cheaper to obtain



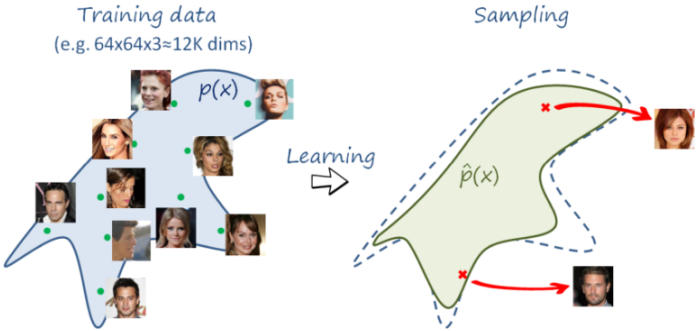
Clustering



Dimensionality Reduction

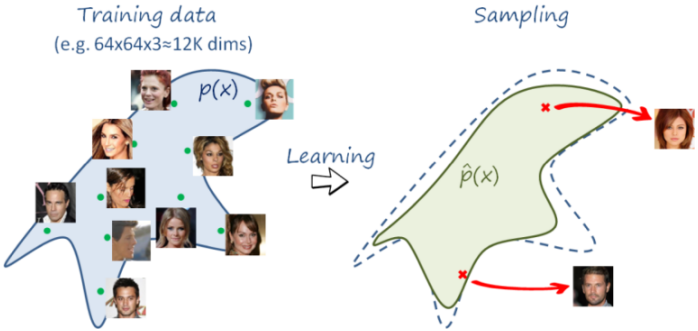
Generative Models

- **Goal**: Generate data samples similar to the ones in the training set



Generative Models

- **Goal**: Generate data samples similar to the ones in the training set



- Assume training data $X = \{x_1, x_2, ..., x_n\}$ comes from an underlying distribution $p_D(x)$, and a generative model samples data from a distribution $p_M(x)$
- Our aim is to **minimize** some notion of **distance** between $p_D(x)$ and $p_M(x)$

Generative Models: How to learn?

Aim: To minimize some notion of distance between $p_D(x)$ and $p_M(x)$; how?

- Given a dataset $X = x_1, x_2, x_3, ..., x_N$ from an underlying distribution $p_D(x)$

Generative Models: How to learn?

Aim: To minimize some notion of distance between $p_D(x)$ and $p_M(x)$; how?

- Given a dataset $X = x_1, x_2, x_3, ..., x_N$ from an underlying distribution $p_D(x)$
- Consider an approximating distribution $p_M(x)$ coming from a family of distributions M, i.e. we need to find the best distribution in M, parametrized by θ, which minimizes distance between p_M and p_D, i.e.:

$$\theta^* = \arg\min_{\theta \in M} \text{dist}(p_\theta, p_D)$$

Generative Models: How to learn?

Aim: To minimize some notion of distance between $p_D(x)$ and $p_M(x)$; how?

- Given a dataset $X = x_1, x_2, x_3, ..., x_N$ from an underlying distribution $p_D(x)$
- Consider an approximating distribution $p_M(x)$ coming from a family of distributions M, i.e. we need to find the best distribution in M, parametrized by θ, which minimizes distance between p_M and p_D, i.e.:

$$\theta^* = \arg\min_{\theta \in M} \text{dist}(p_\theta, p_D)$$

- What distance to choose?

Generative Models: How to learn?

Aim: To minimize some notion of distance between $p_D(x)$ and $p_M(x)$; how?

- Given a dataset $X = x_1, x_2, x_3, ..., x_N$ from an underlying distribution $p_D(x)$
- Consider an approximating distribution $p_M(x)$ coming from a family of distributions M, i.e. we need to find the best distribution in M, parametrized by θ, which minimizes distance between p_M and p_D, i.e.:

$$\theta^* = \arg\min_{\theta \in M} \text{dist}(p_\theta, p_D)$$

- What distance to choose?
- If KL-divergence is the distance function, the above problem becomes one of maximum likelihood estimation!

$$\theta^* = \arg\min_{\theta \in M} \mathbb{E}_{x \sim p_D}[-\log\ p_\theta(x)]$$

Generative Models: How to learn?

Aim: To minimize some notion of distance between $p_D(x)$ and $p_M(x)$; how?

- Given a dataset $X = x_1, x_2, x_3, ..., x_N$ from an underlying distribution $p_D(x)$
- Consider an approximating distribution $p_M(x)$ coming from a family of distributions M, i.e. we need to find the best distribution in M, parametrized by θ, which minimizes distance between p_M and p_D, i.e.:

$$\theta^* = \arg\min_{\theta \in M} \text{dist}(p_\theta, p_D)$$

- What distance to choose?
- If KL-divergence is the distance function, the above problem becomes one of maximum likelihood estimation!

$$\theta^* = \arg\min_{\theta \in M} \mathbb{E}_{x \sim p_D}[-\log\ p_\theta(x)]$$

  Why? Homework!
- This is the idea in a few methods (e.g. PixelCNN/PixelRNN, which we will see later)

Generative Models: Applications

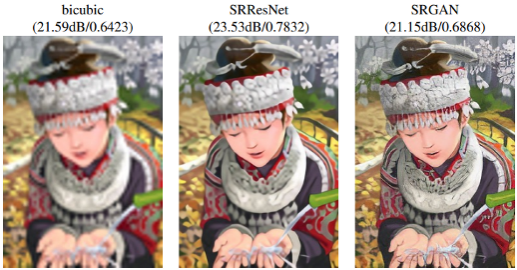
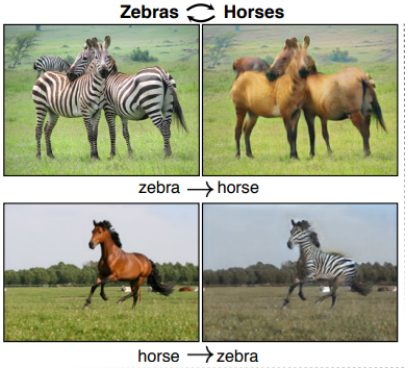


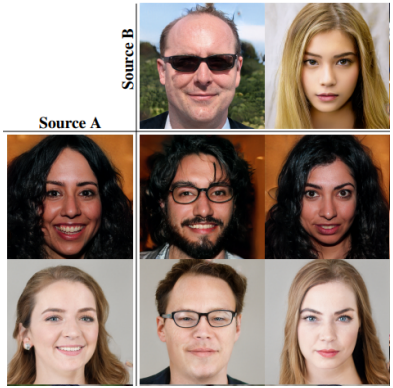
Image Super Resolution

Image Colorization

Generative Models: Applications



Cross-domain Image Translation



Generating Realistic Face Datasets

Generative Models: More Applications

- Learn good generalizable latent features

Generative Models: More Applications

- Learn good generalizable latent features
- Augment small datasets

Generative Models: More Applications

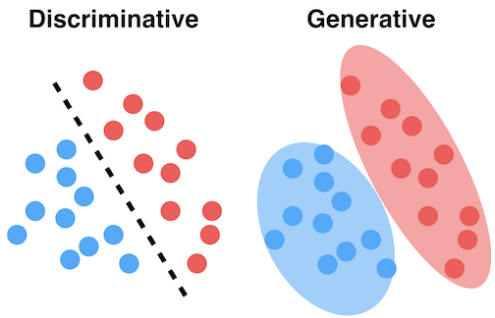
- Learn good generalizable latent features
- Augment small datasets
- Enable mixed-reality applications such as Virtual Try-on

Generative Models: More Applications

- Learn good generalizable latent features
- Augment small datasets
- Enable mixed-reality applications such as Virtual Try-on
- Many more...

What are Generative Models

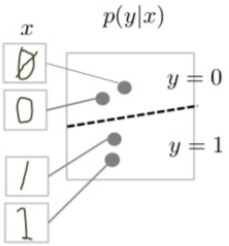
- **Discriminative** models: aim to learn **differentiating features** between various classes in a dataset
- **Generative** models aim to learn **underlying distribution** of each class in a dataset



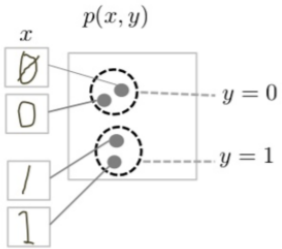
Discriminative vs Generative Models

- Consider a binary classification problem of classifying images of 1s and 0s

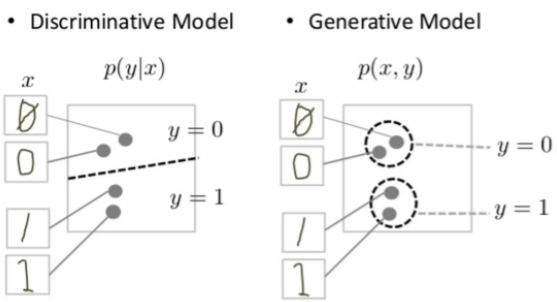
- Discriminative Model



- Generative Model

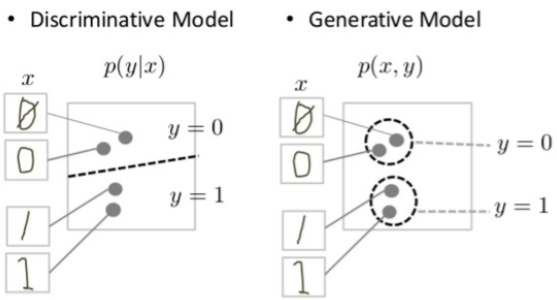


Discriminative vs Generative Models



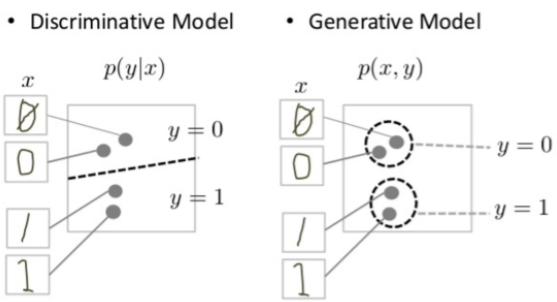
- Consider a binary classification problem of classifying images of 1s and 0s
- A **discriminative classifier** directly models the **posterior** i.e. $p(y|x)$; x is always given as input

Discriminative vs Generative Models



- Consider a binary classification problem of classifying images of 1s and 0s
- A **discriminative classifier** directly models the **posterior** i.e. $p(y|x)$; x is always given as input
- A **generative classifier** models the **joint distribution** i.e. $p(x, y)$

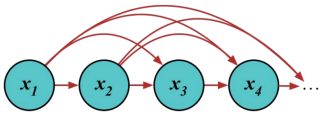
Discriminative vs Generative Models



- Consider a binary classification problem of classifying images of 1s and 0s
- A **discriminative classifier** directly models the **posterior** i.e. $p(y|x)$; x is always given as input
- A **generative classifier** models the **joint distribution** i.e. $p(x, y)$
- Recall: posterior and joint are related as:

$$p(y|x) = \frac{p(x, y)}{p(x)} = \frac{p(x|y)p(y)}{p(x)}$$

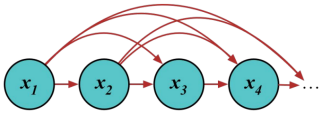
Generative Models

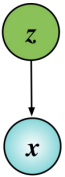


Two main kinds of generative models:

1. **Fully Visible Models:** Directly model observations without introducing extra variables; e.g. considering each pixel value of an image as an observation

Generative Models

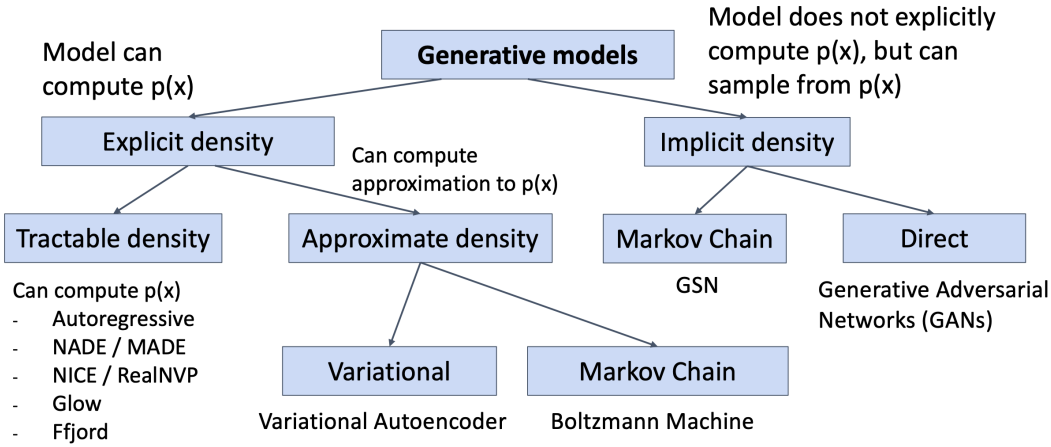




Two main kinds of generative models:

1. **Fully Visible Models:** Directly model observations without introducing extra variables; e.g. considering each pixel value of an image as an observation
2. **Latent Variable Models:** Defining hidden variables which generate observed data:
   1. **Explicit Likelihood Estimation Models:** Explicitly define and learn likelihood of data; e.g. Variational Autoencoders (VAEs)
   2. **Implicit Models:** Learn to directly generate samples from model's distribution, without explicitly defining any density function; e.g. Generative Adversarial Networks (GANs)

Taxonomy of Generative Models



Credit: Fei-Fei Li, J Johnson, CS231n, Stanford Univ

Homework

Readings

- Aditya Grover, Stefano Ermon, Tutorial on Deep Generative Models, IJCAI-ECAI 2018
- (Optional) Shakir Mohamed, Danilo Rezende, Deep Generative Models Tutorial, UAI 2017

Exercise

- Why does using KL-divergence in finding the generative model simplify to maximum likelihood estimation?

Deep Learning for Computer Vision

Generative Adversarial Networks

Vineeth N Balasubramanian

Department of Computer Science and Engineering
Indian Institute of Technology, Hyderabad

Review: Question

Why does using KL-divergence in finding the generative model simplify to maximum likelihood estimation?

- Recall our problem:

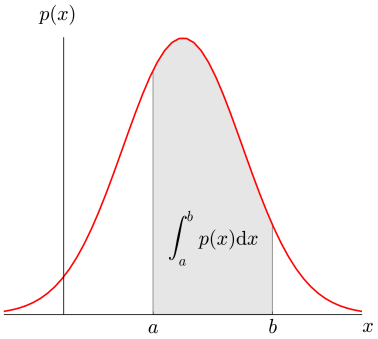
$$\theta^* = \arg\min_{\theta \in M} \text{dist}(p_\theta, p_D)$$

- If distance is KL-divergence:

$$\begin{aligned}
\theta^* &= \arg\min_{\theta \in M} D_{KL}(p_D || p_\theta) \\
&= \arg\min_{\theta \in M} \mathbb{E}_{\mathbf{x} \sim p_D} \left(\log p_D(\mathbf{x}) - \log p_\theta(\mathbf{x}) \right) \\
&= \arg\min_{\theta \in M} \mathbb{E}_{\mathbf{x} \sim p_D} \left(- \log p_\theta(\mathbf{x}) \right) \qquad (\theta \text{ parameters are independent of } \log p_D(\mathbf{x}))
\end{aligned}$$

This is maximum likelihood estimation (minimizing negative log likelihood)!

Recap



- **Generative Models**: Learn probability density function $p(x)$ over images x
- $p(x)$ assigns positive number to each possible image x

  $\implies$ how likely is image x under distribution $p(x)$
- **Applications:**
  - Sample to generate new data
  - Likelihood estimation(Outlier detection)
  - Feature learning (unsupervised)

Density Estimation

Explicit Density Estimation

- Write explicit function $p(x) = f(x, \theta)$
  Input: Image x
  Output: Likelihood value for image
  Parameter: Weights θ
- Assign explicit likelihood to images
- Can enable outlier detection, but compromise generated image quality/sampling speed

Implicit Density Estimation

- Give up estimating explicit form of $p(x)$
- Aim only to sample images from model; no explicit likelihood assignment
- High quality samples generated/fast sampling speed

Generative Adversarial Networks (GANs)[1]

Goal:

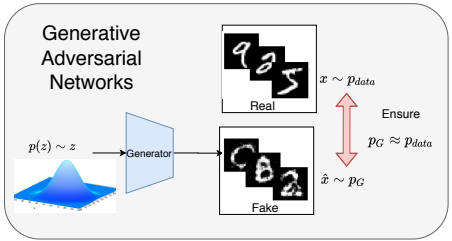
- Build good sampler that allows to draw high quality samples from $p_{model}(x)$
- Not explicitly compute form of $p(x)$ but ensures $p_{model}(x)$ approximates/is close to $p_{data}(x)$
- Output samples similar but not exactly same as train data; how?

[1] Goodfellow et al, Generative Adversarial Nets, NeurIPS 2014

Generative Adversarial Networks (GANs)[1]

Goal:

- Build good sampler that allows to draw high quality samples from $p_{model}(x)$
- Not explicitly compute form of $p(x)$ but ensures $p_{model}(x)$ approximates/is close to $p_{data}(x)$
- Output samples similar but not exactly same as train data; how?



Method:

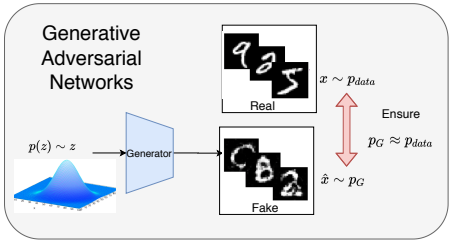
- Introduce latent variable z with simple prior $p(z)$ (e.g. Gaussian)

[1] Goodfellow et al, Generative Adversarial Nets, NeurIPS 2014

Generative Adversarial Networks (GANs)[1]

Goal:

- Build good sampler that allows to draw high quality samples from $p_{model}(x)$
- Not explicitly compute form of $p(x)$ but ensures $p_{model}(x)$ approximates/is close to $p_{data}(x)$
- Output samples similar but not exactly same as train data; how?



Method:

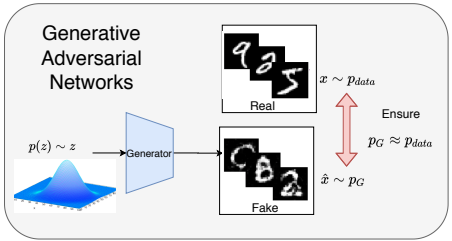
- Introduce latent variable z with simple prior $p(z)$ (e.g. Gaussian)
- Sample $z \sim p(z)$, pass it through Generator $\hat{x} = G(z)$; where $\hat{x} \sim p_G$

[1] Goodfellow et al, Generative Adversarial Nets, NeurIPS 2014

Generative Adversarial Networks (GANs)[1]

Goal:

- Build good sampler that allows to draw high quality samples from $p_{model}(x)$
- Not explicitly compute form of $p(x)$ but ensures $p_{model}(x)$ approximates/is close to $p_{data}(x)$
- Output samples similar but not exactly same as train data; how?

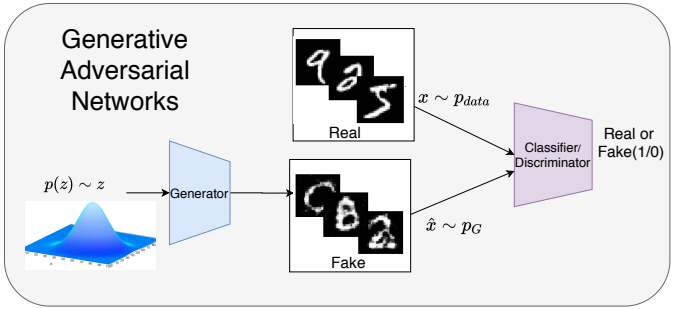


Method:

- Introduce latent variable z with simple prior $p(z)$ (e.g. Gaussian)
- Sample $z \sim p(z)$, pass it through Generator $\hat{x} = G(z)$; where $\hat{x} \sim p_G$
- Introduce mechanism to ensure $p_G \approx p_{data}$

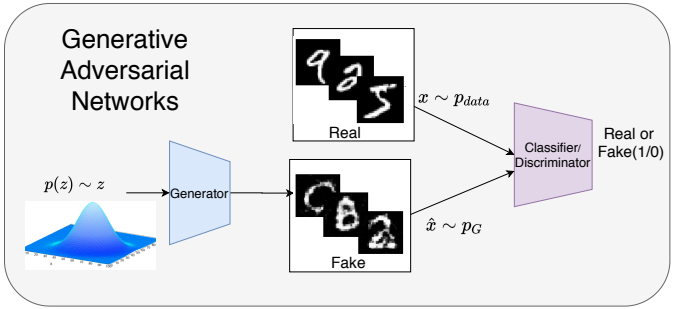
[1] Goodfellow et al, Generative Adversarial Nets, NeurIPS 2014

GANs: Ensuring $p_G \approx p_{data}$



- **Idea**: Use a classifier (called **discriminator**) that differentiates between real samples $x \sim p_{data}$ (class 1) and $\hat{x} \sim p_G$ (class 0)

GANs: Ensuring $p_G \approx p_{data}$



- **Idea**: Use a classifier (called **discriminator**) that differentiates between real samples $x \sim p_{data}$ (class 1) and $\hat{x} \sim p_G$ (class 0)
- Train generator G such that discriminator misclassifies generated sample $\hat{x}$ into class 1
  $\implies$ can no more differentiate between $x \sim p_{data}$ and $\hat{x} \sim p_G$

GANs: How to train?

Training Objective: $\min_G \max_D \left(\mathbb{E}_{x \sim p_{data}}[\log D(x)] + \mathbb{E}_{z \sim p_z}[\log(1 - D(G(z))]\right)$

(**zero-sum game**)

(Assume sigmoid activation in discriminator $\implies$ $D(.) =$ probability that input sample is real)

GANs: How to train?

Training Objective: $\min_G \max_D \left(\mathbb{E}_{x \sim p_{data}}[\log D(x)] + \mathbb{E}_{z \sim p_z}[\log(1 - D(G(z))]\right)$

(**zero-sum game**)

(Assume sigmoid activation in discriminator $\implies$ $D(.) =$ probability that input sample is real)

$$O_1 = \max_D \mathbb{E}_{x \sim p_{data}}[\log D(x)]$$

Train discriminator, D, such that if sample belongs to p_{data}, maximize the log probability of it being a real sample

GANs: How to train?

Training Objective: $\min_G \max_D \left(\mathbb{E}_{x \sim p_{data}}[\log D(x)] + \mathbb{E}_{z \sim p_z}[\log(1 - D(G(z))]\right)$

(**zero-sum game**)

(Assume sigmoid activation in discriminator $\implies$ $D(.) =$ probability that input sample is real)

$$O_1 = \max_D \mathbb{E}_{x \sim p_{data}}[\log D(x)]$$

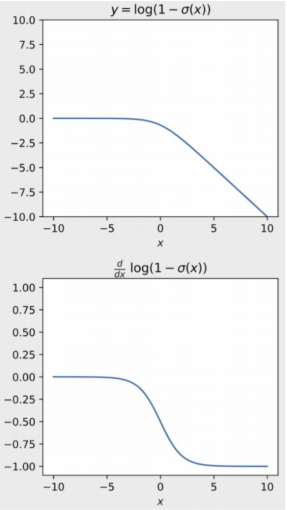
Train discriminator, D, such that if sample belongs to p_{data}, maximize the log probability of it being a real sample

$$O_2 = \min_G \mathbb{E}_{z \sim p_z}[\log(1 - D(G(z))]$$

Train Generator, G such that if sample belongs to p_G(i.e $G(z)$), maximize the log probability of it being a real sample

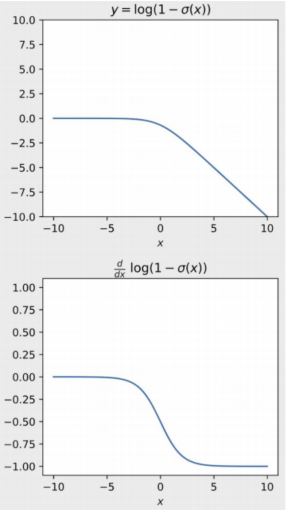
Note: the expectation in the objective function simply implies that losses are averaged over a batch of samples

GANs: Training Strategy



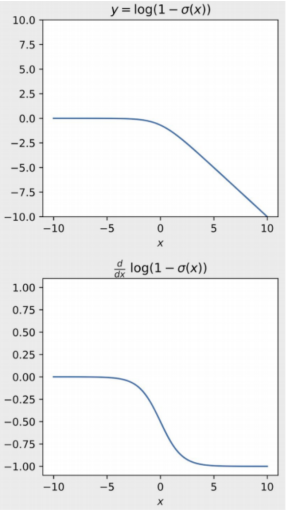
- **Idea:** First train D completely to optimize O_1; then train G to optimize O_2

GANs: Training Strategy



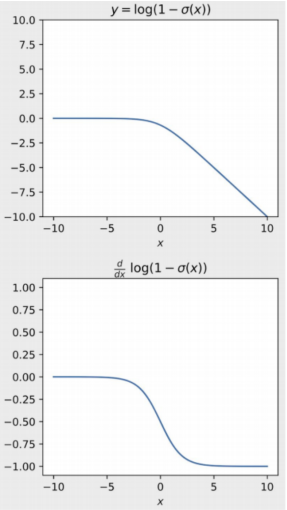
- **Idea:** First train D completely to optimize O_1; then train G to optimize O_2
- **Problem:** If D is initially very confident that samples from G are fake, when x is obtained from G: $D(x)$ or $\sigma(x) = 0 \implies \log(1 - \sigma(x)) = 0$

GANs: Training Strategy



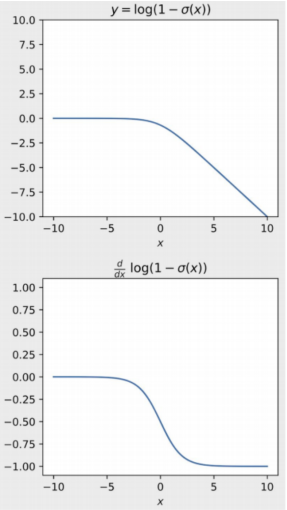
- **Idea:** First train D completely to optimize O_1; then train G to optimize O_2
- **Problem:** If D is initially very confident that samples from G are fake, when x is obtained from G:
  $D(x)$ or $\sigma(x) = 0 \implies \log(1 - \sigma(x)) = 0$
  $\implies \frac{\partial(\log(1-\sigma(x))}{\partial x} = 0$ (as shown in graph)

GANs: Training Strategy



- **Idea:** First train D completely to optimize O_1; then train G to optimize O_2
- **Problem:** If D is initially very confident that samples from G are fake, when x is obtained from G:
  $D(x)$ or $\sigma(x) = 0 \implies \log(1 - \sigma(x)) = 0$
  $\implies \frac{\partial(\log(1-\sigma(x))}{\partial x} = 0$ (as shown in graph)
  $\implies$ G gets no gradients to train!

GANs: Training Strategy



- **Idea:** First train D completely to optimize O_1; then train G to optimize O_2
- **Problem:** If D is initially very confident that samples from G are fake, when x is obtained from G:
  $D(x)$ or $\sigma(x) = 0 \implies \log(1 - \sigma(x)) = 0$
  $\implies \frac{\partial(\log(1-\sigma(x))}{\partial x} = 0$ (as shown in graph)
  $\implies$ G gets no gradients to train!
- **Solution:** Alternate between Discriminator objective O_1 and Generator objective O_2

GANs: Algorithm[2]

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.

[2] Goodfellow et al, Generative Adversarial Nets, NeurIPS 2014

GANs: Evaluating Optimality

Objective: $\min_G \max_D \quad \mathbb{E}_{x \sim p_{data}}[\log D(x)] + \mathbb{E}_{z \sim p_z}[\log(1 - D(G(z))]$

GANs: Evaluating Optimality

Objective: $\min_G \max_D \quad \mathbb{E}_{x \sim p_{data}}[\log D(x)] + \mathbb{E}_{z \sim p_z}[\log(1 - D(G(z))]$

$$\implies \min_G \max_D \int_x \big(p_{data}(x) \log D(x) + p_G(x) \log(1 - D(x)\big) dx$$

(change of variable, expanding expectation)

GANs: Evaluating Optimality

Objective: $\min_G \max_D \quad \mathbb{E}_{x\sim p_{data}}[\log D(x)] + \mathbb{E}_{z\sim p_z}[\log(1 - D(G(z))]$

$$\implies \min_G \max_D \int_x \big(p_{data}(x)\log D(x) + p_G(x)\log(1 - D(x)\big)dx$$

(change of variable, expanding expectation)

$$\implies \min_G \int_x \max_D \big(p_{data}(x)\log D(x) + p_G(x)\log(1 - D(x)\big)dx$$ **(taking $\max$ inside)**

GANs: Evaluating Optimality

Objective: $\min_G \max_D \quad \mathbb{E}_{x \sim p_{data}}[\log D(x)] + \mathbb{E}_{z \sim p_z}[\log(1 - D(G(z))]$

$$\implies \min_G \max_D \int_x \big(p_{data}(x) \log D(x) + p_G(x) \log(1 - D(x)\big) dx$$

(change of variable, expanding expectation)

$$\implies \min_G \int_x \max_D \big(p_{data}(x) \log D(x) + p_G(x) \log(1 - D(x)\big) dx$$ **(taking** $\max$ **inside)**

Let $y = D(x); a = p_{data}; b = p_G$

GANs: Evaluating Optimality

Objective: $\min_G \max_D \quad \mathbb{E}_{x\sim p_{data}}[\log D(x)] + \mathbb{E}_{z\sim p_z}[\log(1 - D(G(z))]$

$$\implies \min_G \max_D \int_x \big(p_{data}(x)\log D(x) + p_G(x)\log(1 - D(x)\big)dx$$

(change of variable, expanding expectation)

$$\implies \min_G \int_x \max_D \big(p_{data}(x)\log D(x) + p_G(x)\log(1 - D(x)\big)dx$$ **(taking $\max$ inside)**

Let $y = D(x); a = p_{data}; b = p_G$

Then, $f(y) = a\log y + b\log(1 - y)$

GANs: Evaluating Optimality

Objective: $\min_G \max_D \quad \mathbb{E}_{x\sim p_{data}}[\log D(x)] + \mathbb{E}_{z\sim p_z}[\log(1 - D(G(z))]$

$$\implies \min_G \max_D \int_x \big(p_{data}(x)\log D(x) + p_G(x)\log(1 - D(x)\big)dx$$

(change of variable, expanding expectation)

$$\implies \min_G \int_x \max_D \big(p_{data}(x)\log D(x) + p_G(x)\log(1 - D(x)\big)dx$$ **(taking $\max$ inside)**

Let $y = D(x); a = p_{data}; b = p_G$

Then, $f(y) = a\log y + b\log(1 - y)$

$$f'(y) = \frac{a}{y} - \frac{b}{1-y}; f'(y) = 0 \implies y = \frac{a}{a+b}$$ **(local max)**

GANs: Evaluating Optimality

Objective: $\min_G \max_D \quad \mathbb{E}_{x\sim p_{data}}[\log D(x)] + \mathbb{E}_{z\sim p_z}[\log(1 - D(G(z))]$

$$\implies \min_G \max_D \int_x \big(p_{data}(x)\log D(x) + p_G(x)\log(1 - D(x)\big)dx$$

(change of variable, expanding expectation)

$$\implies \min_G \int_x \max_D \big(p_{data}(x)\log D(x) + p_G(x)\log(1 - D(x)\big)dx$$ **(taking $\max$ inside)**

Let $y = D(x); a = p_{data}; b = p_G$

Then, $f(y) = a\log y + b\log(1 - y)$

$$f'(y) = \frac{a}{y} - \frac{b}{1-y}; f'(y) = 0 \implies y = \frac{a}{a+b}$$ **(local max)**

Optimal Discriminator: $D^*_G(x) = \frac{p_{data}(x)}{p_{data}(x)+p_G(x)}$

GANs: Evaluating Optimality

$$\min_G \int_X (p_{data}(x) \log D_G^*(x) + p_G(x) \log(1 - D_G^*(x))dx$$

GANs: Evaluating Optimality

$$\min_G \int_X (p_{data}(x) \log D_G^*(x) + p_G(x) \log(1 - D_G^*(x))dx$$

$$\min_G \int_X (p_{data}(x) \left[\log \frac{p_{data}(x)}{p_{data}(x) + p_G(x)} \right] + p_G(x) \left[\log(1 - \frac{p_{data}(x)}{p_{data}(x) + p_G(x)} \right])dx$$

GANs: Evaluating Optimality

$$\min_G \int_X (p_{data}(x) \log D^*_G(x) + p_G(x) \log(1 - D^*_G(x))dx$$

$$\min_G \int_X (p_{data}(x) \left[\log \frac{p_{data}(x)}{p_{data}(x) + p_G(x)} \right] + p_G(x) \left[\log(1 - \frac{p_{data}(x)}{p_{data}(x) + p_G(x)} \right])dx$$

$$\min_G \int_X (p_{data}(x) \left[\log \frac{p_{data}(x)}{p_{data}(x) + p_G(x)} \right] + p_G(x) \left[\log(\frac{p_G(x)}{p_{data}(x) + p_G(x)} \right])dx$$

GANs: Evaluating Optimality

$$\min_G \int_X (p_{data}(x) \log D_G^*(x) + p_G(x) \log(1 - D_G^*(x))dx$$

$$\min_G \int_X (p_{data}(x) \left[\log \frac{p_{data}(x)}{p_{data}(x) + p_G(x)} \right] + p_G(x) \left[\log(1 - \frac{p_{data}(x)}{p_{data}(x) + p_G(x)} \right])dx$$

$$\min_G \int_X (p_{data}(x) \left[\log \frac{p_{data}(x)}{p_{data}(x) + p_G(x)} \right] + p_G(x) \left[\log(\frac{p_G(x)}{p_{data}(x) + p_G(x)} \right])dx$$

$$\min_G \left(\mathbb{E}_{x \sim p_{data}} [\log \frac{p_{data}(x)}{p_{data}(x) + p_G(x)}] + \mathbb{E}_{x \sim p_G} [\log(\frac{p_G(x)}{p_{data}(x) + p_G(x)}] \right)$$

GANs: Evaluating Optimality

$$\min_G \left(\mathbb{E}_{x \sim p_{data}} [\log \frac{2 * p_{data}(x)}{2 * (p_{data}(x) + p_G(x))}] + \mathbb{E}_{x \sim p_G} [\log(\frac{2 * p_G(x)}{2 * (p_{data}(x) + p_G(x))}] \right)$$

GANs: Evaluating Optimality

$$\min_G \left(\mathbb{E}_{x \sim p_{data}} [\log \frac{2 * p_{data}(x)}{2 * (p_{data}(x) + p_G(x))}] + \mathbb{E}_{x \sim p_G} [\log (\frac{2 * p_G(x)}{2 * (p_{data}(x) + p_G(x))}] \right)$$

$$\min_G \left(\mathbb{E}_{x \sim p_{data}} [\log \frac{2 * p_{data}(x)}{p_{data}(x) + p_G(x)}] + \mathbb{E}_{x \sim p_G} [\log (\frac{2 * p_G(x)}{p_{data}(x) + p_G(x)}] - \log 4 \right)$$

GANs: Evaluating Optimality

$$\min_G \left(\mathbb{E}_{x \sim p_{data}}[\log \frac{2 * p_{data}(x)}{2 * (p_{data}(x) + p_G(x))}] + \mathbb{E}_{x \sim p_G}[\log(\frac{2 * p_G(x)}{2 * (p_{data}(x) + p_G(x))}] \right)$$

$$\min_G \left(\mathbb{E}_{x \sim p_{data}}[\log \frac{2 * p_{data}(x)}{p_{data}(x) + p_G(x)}] + \mathbb{E}_{x \sim p_G}[\log(\frac{2 * p_G(x)}{p_{data}(x) + p_G(x)}] - \log 4 \right)$$

$$\min_G \left(\mathbf{KL}(p_{data}(x), \frac{p_{data}(x) + p_G(x)}{2}) + \mathbf{KL}(p_G(x), \frac{p_{data}(x) + p_G(x)}{2}) - \log 4 \right)$$

GANs: Evaluating Optimality

$$\min_G \left(\mathbb{E}_{x \sim p_{data}}[\log \frac{2 * p_{data}(x)}{2 * (p_{data}(x) + p_G(x))}] + \mathbb{E}_{x \sim p_G}[\log(\frac{2 * p_G(x)}{2 * (p_{data}(x) + p_G(x))}] \right)$$

$$\min_G \left(\mathbb{E}_{x \sim p_{data}}[\log \frac{2 * p_{data}(x)}{p_{data}(x) + p_G(x)}] + \mathbb{E}_{x \sim p_G}[\log(\frac{2 * p_G(x)}{p_{data}(x) + p_G(x)}] - \log 4 \right)$$

$$\min_G \left(\mathbf{KL}(p_{data}(x), \frac{p_{data}(x) + p_G(x)}{2}) + \mathbf{KL}(p_G(x), \frac{p_{data}(x) + p_G(x)}{2}) - \log 4 \right)$$

Kullback–Leibler Divergence

$\mathbf{KL(p,q)} = \mathbb{E}_{x \sim p}[\frac{\log p(x)}{\log q(x)}]$

Jenson-Shannon Divergence

$\mathbf{JSD}(p_{data}, p_G) = \frac{KL(p_{data}, \frac{p_{data} + p_G}{2})}{2} + \frac{KL(p_G, \frac{p_{data} + p_G}{2})}{2}$

$\min_G(2 * JSD(p_{data}, p_G) - \log 4) \implies$ expression is minimized when $p_{data} = p_G$

(Recall $JSD \geq 0$ by definition)

Optimality of GANs: Conclusions

- $D^*_G(x) = \frac{p_{data}(x)}{p_{data}(x)+p_G(x)}$ (Optimal Discriminator for any G)

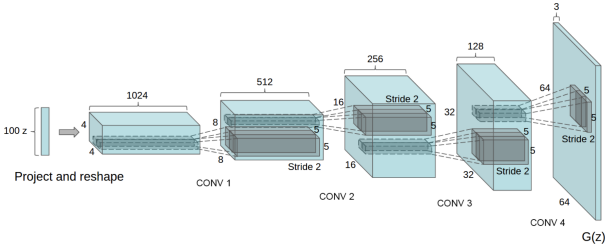
Optimality of GANs: Conclusions

- $D^*_G(x) = \frac{p_{data}(x)}{p_{data}(x)+p_G(x)}$ (Optimal Discriminator for any G)
- $p_{data}(x) = p_G(x)$ (Optimal Generator for any D)

Optimality of GANs: Conclusions

- $D^*_G(x) = \frac{p_{data}(x)}{p_{data}(x)+p_G(x)}$ (Optimal Discriminator for any G)
- $p_{data}(x) = p_G(x)$ (Optimal Generator for any D)
- $D^*_G(x) = \frac{p_G(x)}{p_G(x)+p_G(x)} = \frac{p_{data}(x)}{p_{data}(x)+p_{data}(x)} = \frac{1}{2}$

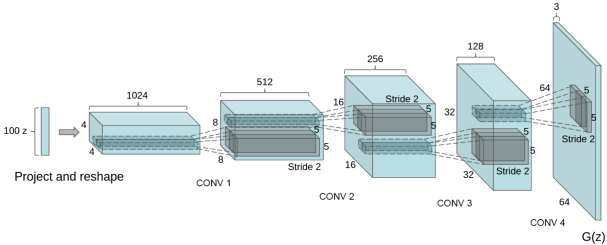
Deep Convolution GAN (DCGAN)[3]



- Bridge the gap between success of CNNs for supervised learning and unsupervised generative models

[3]Radford et al, Unsupervised Representation learning with deep convolutional GANs, NeurIPS 2016

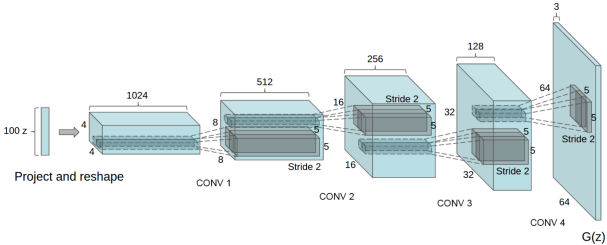
Deep Convolution GAN (DCGAN)[3]



- Bridge the gap between success of CNNs for supervised learning and unsupervised generative models
- Best practices to enable stable training of GANs with deep architectures

[3]Radford et al, Unsupervised Representation learning with deep convolutional GANs, NeurIPS 2016

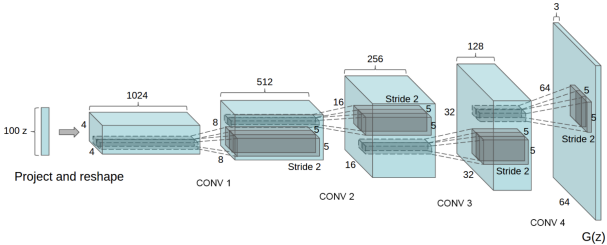
Deep Convolution GAN (DCGAN)[3]



- Bridge the gap between success of CNNs for supervised learning and unsupervised generative models
- Best practices to enable stable training of GANs with deep architectures
- Smooth high-dimensional Interpolations; Vector arithmetic

[3]Radford et al, Unsupervised Representation learning with deep convolutional GANs, NeurIPS 2016

Deep Convolution GAN (DCGAN)[3]



- Bridge the gap between success of CNNs for supervised learning and unsupervised generative models
- Best practices to enable stable training of GANs with deep architectures
- Smooth high-dimensional Interpolations; Vector arithmetic
- Demonstrate unsupervised feature learning capabilities of GANs and its applications **(Representation Learning)**

[3] Radford et al, Unsupervised Representation learning with deep convolutional GANs, NeurIPS 2016

DCGAN: Training Practices for Deeper GANs[4]

- Replaces deterministic spatial pooling functions (e.g maxpooling) with **strided convolutions** $\rightarrow$ allows to learn spatial downsampling

[4] Radford et al, Unsupervised Representation Learning with Deep Convolutional GANs, NeurIPS 2016

DCGAN: Training Practices for Deeper GANs[4]

- Replaces deterministic spatial pooling functions (e.g maxpooling) with **strided convolutions** $\rightarrow$ allows to learn spatial downsampling
- **Remove fully connected hidden layers** for deeper architectures

[4] Radford et al, Unsupervised Representation Learning with Deep Convolutional GANs, NeurIPS 2016

DCGAN: Training Practices for Deeper GANs[4]

- Replaces deterministic spatial pooling functions (e.g maxpooling) with **strided convolutions** $\rightarrow$ allows to learn spatial downsampling
- **Remove fully connected hidden layers** for deeper architectures
- **Batchnorm in G and D** $\rightarrow$ prevent generator collapse/enable gradient flow in deeper architectures; not applied at output of G and input of D

[4] Radford et al, Unsupervised Representation Learning with Deep Convolutional GANs, NeurIPS 2016

DCGAN: Training Practices for Deeper GANs[4]

- Replaces deterministic spatial pooling functions (e.g maxpooling) with **strided convolutions** $\rightarrow$ allows to learn spatial downsampling
- **Remove fully connected hidden layers** for deeper architectures
- **Batchnorm in G and D** $\rightarrow$ prevent generator collapse/enable gradient flow in deeper architectures; not applied at output of G and input of D
- **Non-Linearity:** `ReLU` for generator, `Leaky-ReLU (0.2)` for discriminator

[4] Radford et al, Unsupervised Representation Learning with Deep Convolutional GANs, NeurIPS 2016

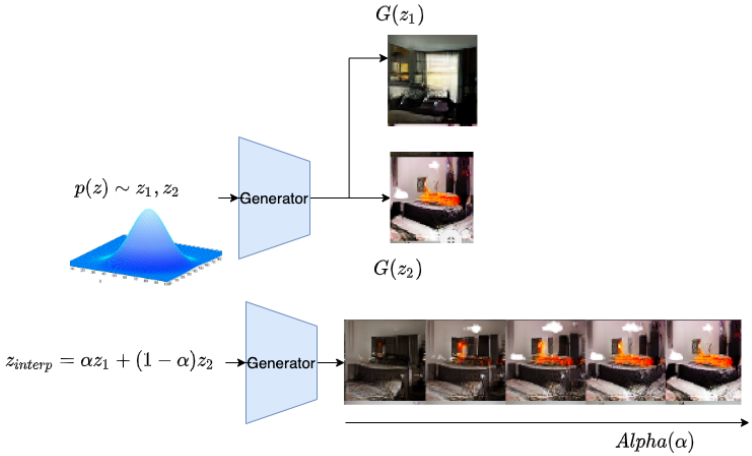
DCGAN: Training Practices for Deeper GANs[4]

- Replaces deterministic spatial pooling functions (e.g maxpooling) with **strided convolutions** $\rightarrow$ allows to learn spatial downsampling
- **Remove fully connected hidden layers** for deeper architectures
- **Batchnorm in G and D** $\rightarrow$ prevent generator collapse/enable gradient flow in deeper architectures; not applied at output of G and input of D
- **Non-Linearity:** `ReLU` for generator, `Leaky-ReLU (0.2)` for discriminator
- **Output Non-Linearity:** `tanh` for generator, `sigmoid` for discriminator

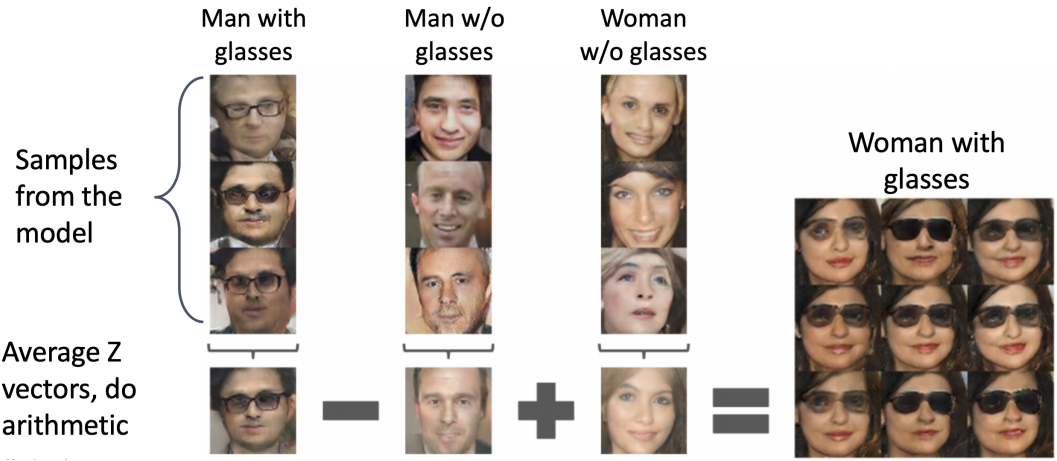
[4] Radford et al, Unsupervised Representation Learning with Deep Convolutional GANs, NeurIPS 2016

Walking the Latent Space

- We can interpolate between two points in latent space and visualize
- Smooth transition in generated image space by changing latent (perceptual) features is a sign of a good model



Vector Arithmetic in Latent Space[5]



[5] Radford et al, Unsupervised Representation Learning with Deep Convolutional GANs, NeurIPS 2016

Pose Transformation[6]

- z_{left}: averaged latent vector of faces looking left
  z_{right}: average latent vector of faces looking right
- $z_{\text{turn}} = z_{\text{right}} - z_{\text{left}}$
  $z_{\text{new}} = z + \alpha z_{\text{turn}}$
- $G(z_{\text{new}}) \Longrightarrow$ Transformed image

[6] Radford et al, Unsupervised Representation Learning with Deep Convolutional GANs, NeurIPS 2016

Feature Learning[7]

Table 1: CIFAR-10 classification results using our pre-trained model. Our DCGAN is not pre-trained on CIFAR-10, but on Imagenet-1k, and the features are used to classify CIFAR-10 images.

Model	Accuracy	Accuracy (400 per class)	max # of features units
1 Layer K-means	80.6%	63.7% (±0.7%)	4800
3 Layer K-means Learned RF	82.0%	70.7% (±0.7%)	3200
View Invariant K-means	81.9%	72.6% (±0.7%)	6400
Exemplar CNN	84.3%	77.4% (±0.2%)	1024
DCGAN (ours) + L2-SVM	82.8%	73.8% (±0.4%)	512

- Train DCGAN on ImageNet-1k

[7] Radford et al, Unsupervised Representation Learning with Deep Convolutional GANs, NeurIPS 2016

Feature Learning[7]

Table 1: CIFAR-10 classification results using our pre-trained model. Our DCGAN is not pre-trained on CIFAR-10, but on Imagenet-1k, and the features are used to classify CIFAR-10 images.

Model	Accuracy	Accuracy (400 per class)	max # of features units
1 Layer K-means	80.6%	63.7% (±0.7%)	4800
3 Layer K-means Learned RF	82.0%	70.7% (±0.7%)	3200
View Invariant K-means	81.9%	72.6% (±0.7%)	6400
Exemplar CNN	84.3%	77.4% (±0.2%)	1024
DCGAN (ours) + L2-SVM	82.8%	73.8% (±0.4%)	512

- Train DCGAN on ImageNet-1k
- Use discriminator's convolution features for supervised tasks i.e. classify images from CIFAR-10 dataset

[7] Radford et al, Unsupervised Representation Learning with Deep Convolutional GANs, NeurIPS 2016

Feature Learning[7]

Table 1: CIFAR-10 classification results using our pre-trained model. Our DCGAN is not pre-trained on CIFAR-10, but on Imagenet-1k, and the features are used to classify CIFAR-10 images.

Model	Accuracy	Accuracy (400 per class)	max # of features units
1 Layer K-means	80.6%	63.7% (±0.7%)	4800
3 Layer K-means Learned RF	82.0%	70.7% (±0.7%)	3200
View Invariant K-means	81.9%	72.6% (±0.7%)	6400
Exemplar CNN	84.3%	77.4% (±0.2%)	1024
DCGAN (ours) + L2-SVM	82.8%	73.8% (±0.4%)	512

- Train DCGAN on ImageNet-1k
- Use discriminator's convolution features for supervised tasks i.e. classify images from CIFAR-10 dataset
- DCGAN never trained on CIFAR-10 $\implies$ representation power and domain robustness of learned features

[7] Radford et al, Unsupervised Representation Learning with Deep Convolutional GANs, NeurIPS 2016

Feature Learning[7]

Table 1: CIFAR-10 classification results using our pre-trained model. Our DCGAN is not pre-trained on CIFAR-10, but on Imagenet-1k, and the features are used to classify CIFAR-10 images.

Model	Accuracy	Accuracy (400 per class)	max # of features units
1 Layer K-means	80.6%	63.7% ($\pm$0.7%)	4800
3 Layer K-means Learned RF	82.0%	70.7% ($\pm$0.7%)	3200
View Invariant K-means	81.9%	72.6% ($\pm$0.7%)	6400
Exemplar CNN	84.3%	77.4% ($\pm$0.2%)	1024
DCGAN (ours) + L2-SVM	82.8%	73.8% ($\pm$0.4%)	512

- Train DCGAN on ImageNet-1k
- Use discriminator's convolution features for supervised tasks i.e. classify images from CIFAR-10 dataset
- DCGAN never trained on CIFAR-10 $\implies$ representation power and domain robustness of learned features
- Competitive results when compared with other feature learning methods

[7] Radford et al, Unsupervised Representation Learning with Deep Convolutional GANs, NeurIPS 2016

GANs: How to evaluate?

- **Human judgement:** Good generator $\implies$ images with distinctly recognizable objects; Semantically diverse samples

GANs: How to evaluate?

- **Human judgement:** Good generator $\implies$ images with distinctly recognizable objects; Semantically diverse samples
  - **Recognizable objects** $\implies$ classifier predicts class of generated sample correctly with high confidence
  - **Semantic diversity** $\implies$ generated samples must be evenly from all classes in train set

GANs: How to evaluate?

- **Human judgement:** Good generator $\implies$ images with distinctly recognizable objects; Semantically diverse samples
  - **Recognizable objects** $\implies$ classifier predicts class of generated sample correctly with high confidence
  - **Semantic diversity** $\implies$ generated samples must be evenly from all classes in train set
- **Prediction power**: How ImageNet pre-trained Inception Network V3 performs on these generated images

GANs: How to evaluate?

- **Human judgement:** Good generator $\implies$ images with distinctly recognizable objects; Semantically diverse samples
  - **Recognizable objects** $\implies$ classifier predicts class of generated sample correctly with high confidence
  - **Semantic diversity** $\implies$ generated samples must be evenly from all classes in train set
- **Prediction power**: How ImageNet pre-trained Inception Network V3 performs on these generated images
- Still an open research problem...

Inception Score[8]

- Correlates with human judgement
- $p(y|x)$: Inception model/classifier
- $p(y)$: Label/class distribution of generated samples

[8] Radford et al, Unsupervised Representation Learning with Deep Convolutional GANs, NeurIPS 2016

Inception Score[8]

- Correlates with human judgement
- $p(y|x)$: Inception model/classifier
- $p(y)$: Label/class distribution of generated samples
- **Inception Score(IS):** $= \exp(\mathbb{E}_{x\sim p_g}[D_{KL}[p(y|x)||p(y)])$

$$= \exp(\mathbb{E}_{x\sim p_g, y\sim p(y|x)}[\log(p(y|x)) - \log(p(y)])$$
$$= \exp(H(y) - H(y|x))$$

$H(y)$: entropy of generated sample class labels (**Semantic diversity = high** $H(y)$)

$H(y/x)$: entropy of class labels from classifier (**Distinctly Classifiable = low** $H(y|x)$)

[8] Radford et al, Unsupervised Representation Learning with Deep Convolutional GANs, NeurIPS 2016

Inception Score[8]

- Correlates with human judgement
- $p(y|x)$: Inception model/classifier
- $p(y)$: Label/class distribution of generated samples
- **Inception Score(IS):** $= \exp(\mathbb{E}_{x\sim p_g}[D_{KL}[p(y|x)||p(y)])$

$$= \exp(\mathbb{E}_{x\sim p_g, y\sim p(y|x)}[\log(p(y|x)) - \log(p(y)])$$
$$= \exp(H(y) - H(y|x))$$

$H(y)$: entropy of generated sample class labels (**Semantic diversity = high** $H(y)$)

$H(y/x)$: entropy of class labels from classifier (**Distinctly Classifiable = low** $H(y|x)$)

- Higher IS $\implies$ better generative model

[8] Radford et al, Unsupervised Representation Learning with Deep Convolutional GANs, NeurIPS 2016

Fréchet Inception Distance (FID)[9]

- **Drawback of IS:** Statistics of real-world samples are not used to compare with statistics of synthetic samples
- FID enables us to capture more subtle differences; measures diversity better

[9] Heusel et al, GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium, NeurIPS 2017

Fréchet Inception Distance (FID)[9]

- **Drawback of IS:** Statistics of real-world samples are not used to compare with statistics of synthetic samples
- FID enables us to capture more subtle differences; measures diversity better
- **Intution:** Distance between real-world data distribution and generating model's data distribution serves as performance measure for generative models; how?

[9] Heusel et al, GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium, NeurIPS 2017

Fréchet Inception Distance (FID)[9]

- **Drawback of IS:** Statistics of real-world samples are not used to compare with statistics of synthetic samples
- FID enables us to capture more subtle differences; measures diversity better
- **Intution:** Distance between real-world data distribution and generating model's data distribution serves as performance measure for generative models; how?
- Embed image x into feature space (activations of Inception-v3 pool3 layer), and compute the Fréchet distance between two multivariate Gaussians:

$$d^2((m, C), (m_w, C_w)) = ||m - m_w||_2^2 + Tr(C + C_w - 2(C \cdot C_w)^2)$$

where m is mean, C is covariance of generated image features; m_w is mean, C_w is covariance of original image features

[9] Heusel et al, GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium, NeurIPS 2017

Fréchet Inception Distance (FID)[9]

- **Drawback of IS:** Statistics of real-world samples are not used to compare with statistics of synthetic samples
- FID enables us to capture more subtle differences; measures diversity better
- **Intution:** Distance between real-world data distribution and generating model's data distribution serves as performance measure for generative models; how?
- Embed image x into feature space (activations of Inception-v3 pool3 layer), and compute the Fréchet distance between two multivariate Gaussians:

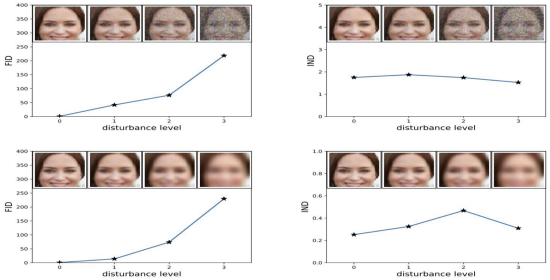
$$d^2((m, C), (m_w, C_w)) = ||m - m_w||_2^2 + Tr(C + C_w - 2(C \cdot C_w)^2)$$

  where m is mean, C is covariance of generated image features; m_w is mean, C_w is covariance of original image features
- Lower FID $\implies$ better generative model

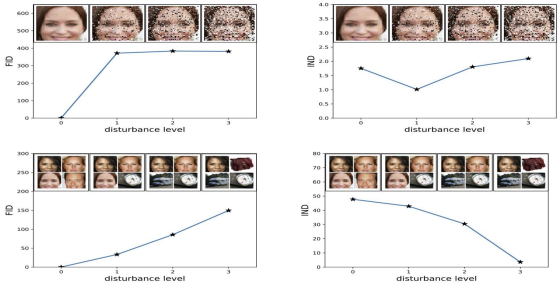
[9] Heusel et al, GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium, NeurIPS 2017

FID vs Inception Score[10]

CelebA dataset (Columns 1,3: FID Score; Columns 2,4: Inception Score)



(Top:) Gaussian noise added to images
(Bottom:) Gaussian blur added to images



(Top:) Salt-and-pepper noise added to images
(Bottom:) ImageNet crops added to images

[10] Heusel et al, GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium, NeurIPS 2017

Homework

Readings

- Dive into Deep Learning, Chapter 17
  https://d2l.ai/chapter_generative-adversarial-networks/gan.html
- Play with Generative Adversarial Networks (GANs) in your browser
  https://poloclub.github.io/ganlab/
- Deep Learning With Python - François Chollet, Section 8.5
  https://github.com/veritone/ds-transcription-capstone

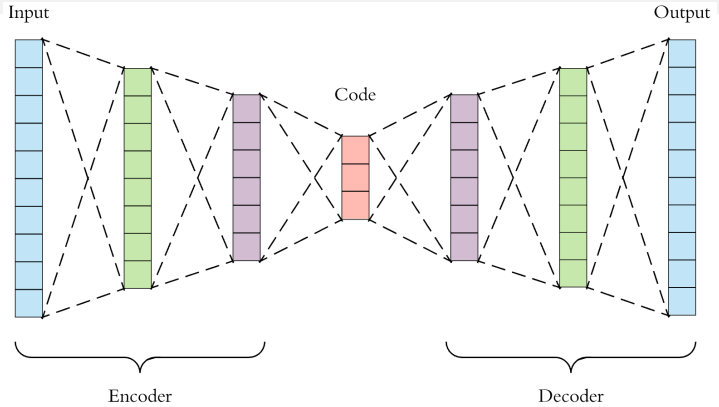
Deep Learning for Computer Vision

Variational Auto-Encoders

Vineeth N Balasubramanian

Department of Computer Science and Engineering
Indian Institute of Technology, Hyderabad

Recall: Autoencoders

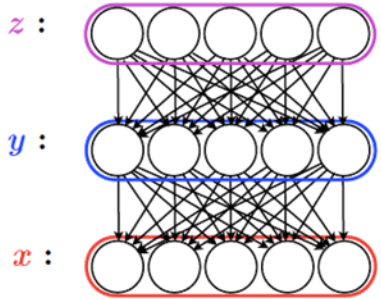


Autoencoders can reconstruct data, and can learn features to initialize a supervised model.
Can we generate images from an autoencoder?

Credit: Arden Dertat, TowardsDataScience

Variational Autoencoders

- Introduced around the same time by two groups of researchers:
  - Kingma and Welling, **Auto-Encoding Variational Bayes**, ICLR 2014
  - Rezende, Mohamed and Wiestra, **Stochastic Backpropagation and Variational Inference in Deep Latent Gaussian Models**, ICML 2014



Credit: Aaron Courville, Deep Learning Summer School, 2015

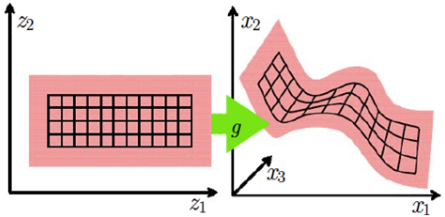
Variational Autoencoders

- **Latent Variable Model:** Learn a mapping from some latent variable z to a possibly complex distribution on x

$$p(x) = \int p(x, z)dz \quad \text{where} \quad p(x, z) = p(x|z)p(z)$$

$$p(z) = \text{something simple}; \quad p(x|z) = g(z)$$

- Can we learn to decouple the true explanatory factors (**latent variables**) underlying the data distribution (e.g. identity and expression in face images)? How?



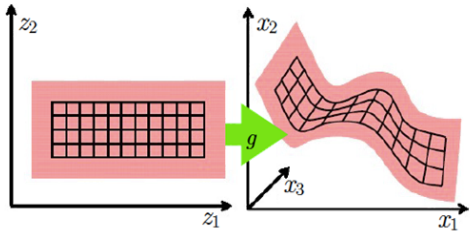
Credit: Aaron Courville, Deep Learning Summer School, 2015

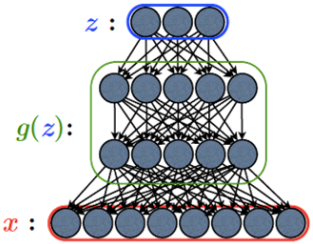
Variational Autoencoders

- Leverage neural networks to learn a latent variable model!

$$p(x) = \int p(x, z)dz \quad \text{where} \quad p(x, z) = p(x|z)p(z)$$

$$p(z) = \text{something simple}; \quad p(x|z) = g(z)$$





Credit: Aaron Courville, Deep Learning Summer School, 2015

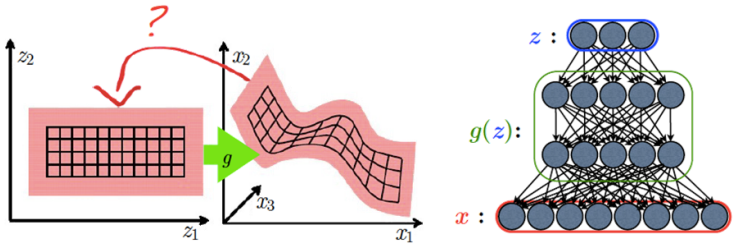
Variational Autoencoders

- Leverage neural networks to learn a latent variable model!

$$p(x) = \int p(x, z)dz \quad \text{where} \quad p(x, z) = p(x|z)p(z)$$

$$p(z) = \text{something simple}; \quad p(x|z) = g(z)$$

- Where does z come from? Computing the posterior $p(z|x)$ is intractable, and we need it to train the directed model

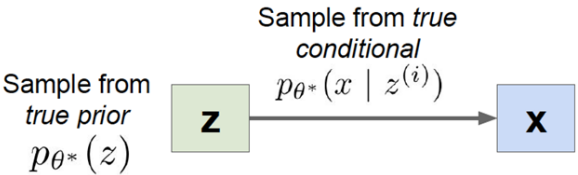


Credit: Aaron Courville, Deep Learning Summer School, 2015

Variational Autoencoders

A Bayesian spin on an autoencoder!

Assume our data $\{x^{(i)}\}_{i=1}^{N}$ is generated like this:



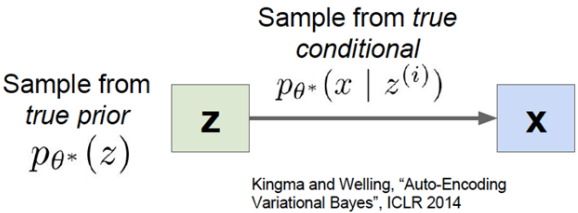
Kingma and Welling, "Auto-Encoding Variational Bayes", ICLR 2014

Credit: Fei-Fei Li, Andrej Karpathy and Justin Johnson, CS231n, Stanford Univ

Variational Autoencoders

A Bayesian spin on an autoencoder!

Assume our data $\{x^{(i)}\}_{i=1}^{N}$ is generated like this:



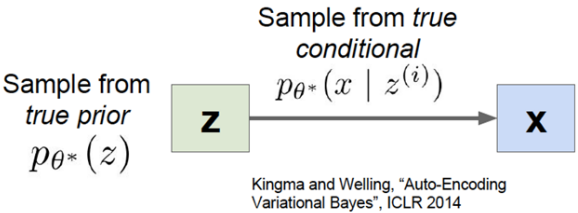
- **Intuition:** x is an image, z gives class, orientation, attributes, etc

Credit: Fei-Fei Li, Andrej Karpathy and Justin Johnson, CS231n, Stanford Univ

Variational Autoencoders

A Bayesian spin on an autoencoder!

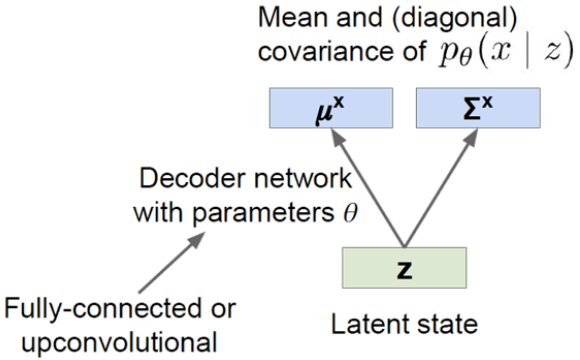
Assume our data $\{x^{(i)}\}_{i=1}^{N}$ is generated like this:



- **Intuition:** x is an image, z gives class, orientation, attributes, etc
- **Problem:** Estimate θ without access to latent states $z^{(i)}$

Credit: Fei-Fei Li, Andrej Karpathy and Justin Johnson, CS231n, Stanford Univ

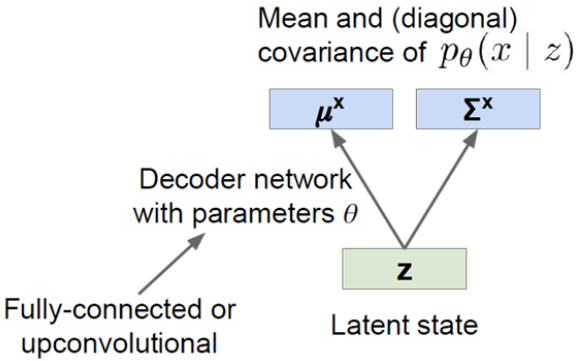
Variational Autoencoders



- **Prior:** Assume $p_\theta(z)$ is a unit Gaussian

Credit: Fei-Fei Li, Andrej Karpathy and Justin Johnson, CS231n, Stanford Univ

Variational Autoencoders



- **Prior:** Assume $p_\theta(z)$ is a unit Gaussian
- **Conditional:** Assume $p_\theta(x|z)$ is a diagonal Gaussian, predict mean and variance with neural network

Credit: Fei-Fei Li, Andrej Karpathy and Justin Johnson, CS231n, Stanford Univ

Variational Autoencoders

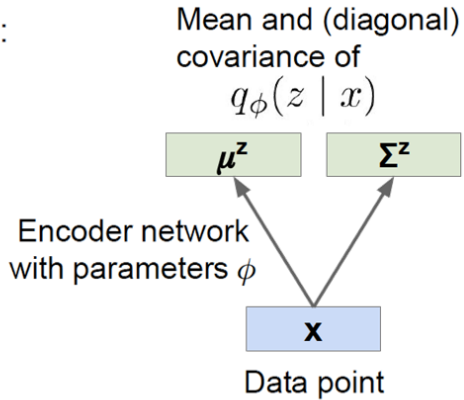
By Bayes Rule the posterior is:

$$p_\theta(z \mid x) = \frac{p_\theta(x \mid z)\, p_\theta(z)}{p_\theta(x)}$$

Use decoder network ☺

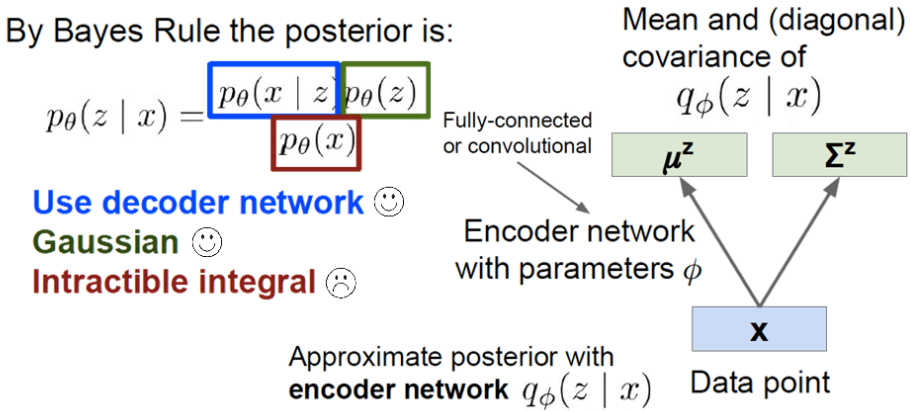
Gaussian ☺

Intractible integral ☹



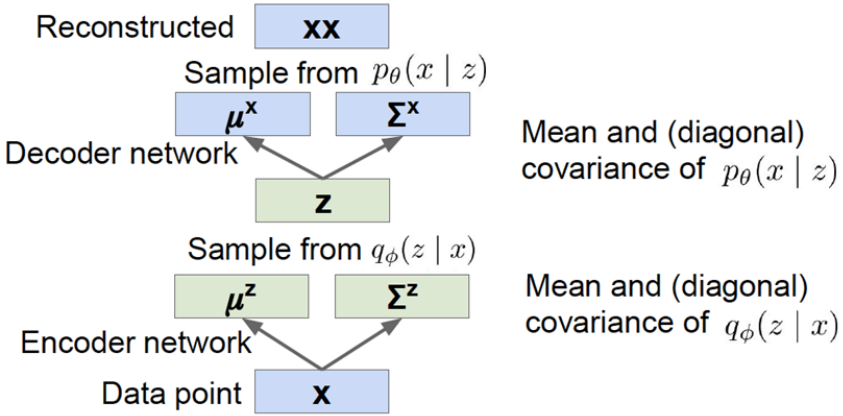
Credit: Fei-Fei Li, Andrej Karpathy and Justin Johnson, CS231n, Stanford Univ

Variational Autoencoders



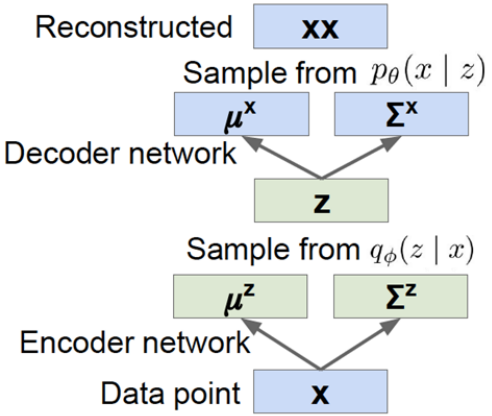
Credit: Fei-Fei Li, Andrej Karpathy and Justin Johnson, CS231n, Stanford Univ

Variational Autoencoders



Credit: Fei-Fei Li, Andrej Karpathy and Justin Johnson, CS231n, Stanford Univ

Variational Autoencoders



Training like a normal autoencoder:
reconstruction loss at the end,
regularization toward prior in middle

Mean and (diagonal)
covariance of $p_\theta(x \mid z)$
(should be close to data x)

Mean and (diagonal)
covariance of $q_\phi(z \mid x)$
(should be close
to prior $p_\theta(z)$)

Credit: Fei-Fei Li, Andrej Karpathy and Justin Johnson, CS231n, Stanford Univ

Variational Autoencoder: The Math

$$\theta^* = \arg\max_\theta \prod_{i=1}^{N} p_\theta(x^{(i)})$$

Maximize likelihood of dataset $\{x^{(i)}\}_{i=1}^{N}$

Credit: Fei-Fei Li, Andrej Karpathy and Justin Johnson, CS231n, Stanford Univ

Variational Autoencoder: The Math

$$\theta^* = \arg\max_\theta \prod_{i=1}^{N} p_\theta(x^{(i)})$$

Maximize likelihood of dataset $\{x^{(i)}\}_{i=1}^{N}$

$$= \arg\max_\theta \sum_{i=1}^{N} \log p_\theta(x^{(i)})$$

Maximize log-likelihood instead because sums are nicer

Credit: Fei-Fei Li, Andrej Karpathy and Justin Johnson, CS231n, Stanford Univ

Variational Autoencoder: The Math

$$\theta^* = \arg\max_\theta \prod_{i=1}^{N} p_\theta(x^{(i)})$$

Maximize likelihood of dataset $\{x^{(i)}\}_{i=1}^{N}$

$$= \arg\max_\theta \sum_{i=1}^{N} \log p_\theta(x^{(i)})$$

Maximize log-likelihood instead because sums are nicer

$$p_\theta(x^{(i)}) = \int p_\theta(x^{(i)}, z) dz$$

Marginalize joint distribution

Credit: Fei-Fei Li, Andrej Karpathy and Justin Johnson, CS231n, Stanford Univ

Variational Autoencoder: The Math

$$\theta^* = \arg\max_\theta \prod_{i=1}^{N} p_\theta(x^{(i)})$$

Maximize likelihood of dataset $\{x^{(i)}\}_{i=1}^N$

$$= \arg\max_\theta \sum_{i=1}^{N} \log p_\theta(x^{(i)})$$

Maximize log-likelihood instead because sums are nicer

$$p_\theta(x^{(i)}) = \int p_\theta(x^{(i)}, z) dz$$

Marginalize joint distribution

$$= \int p_\theta(x^{(i)} \mid z) p_\theta(z) dz$$

Intractible integral ☹

Credit: Fei-Fei Li, Andrej Karpathy and Justin Johnson, CS231n, Stanford Univ

Variational Autoencoder: The Math

$$
\begin{aligned}
\log p_\theta(x^{(i)}) &= \mathbf{E}_{z\sim q_\phi(z|x^{(i)})}\left[\log p_\theta(x^{(i)})\right] && (p_\theta(x^{(i)}) \text{ Does not depend on } z) \\
&= \mathbf{E}_z\left[\log \frac{p_\theta(x^{(i)} \mid z)p_\theta(z)}{p_\theta(z \mid x^{(i)})}\right] && \text{(Bayes' Rule)} \\
&= \mathbf{E}_z\left[\log \frac{p_\theta(x^{(i)} \mid z)p_\theta(z)}{p_\theta(z \mid x^{(i)})}\frac{q_\phi(z \mid x^{(i)})}{q_\phi(z \mid x^{(i)})}\right] && \text{(Multiply by constant)}
\end{aligned}
$$

Credit: Fei-Fei Li, Andrej Karpathy and Justin Johnson, CS231n, Stanford Univ

Variational Autoencoder: The Math

$$
\begin{aligned}
\log p_\theta(x^{(i)}) &= \mathbf{E}_{z\sim q_\phi(z|x^{(i)})}\left[\log p_\theta(x^{(i)})\right] && (p_\theta(x^{(i)}) \text{ Does not depend on } z)\\
&= \mathbf{E}_z\left[\log \frac{p_\theta(x^{(i)} \mid z)p_\theta(z)}{p_\theta(z \mid x^{(i)})}\right] && \text{(Bayes' Rule)}\\
&= \mathbf{E}_z\left[\log \frac{p_\theta(x^{(i)} \mid z)p_\theta(z)}{p_\theta(z \mid x^{(i)})}\frac{q_\phi(z \mid x^{(i)})}{q_\phi(z \mid x^{(i)})}\right] && \text{(Multiply by constant)}\\
&= \mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - \mathbf{E}_z\left[\log \frac{q_\phi(z \mid x^{(i)})}{p_\theta(z)}\right] + \mathbf{E}_z\left[\log \frac{q_\phi(z \mid x^{(i)})}{p_\theta(z \mid x^{(i)})}\right] && \text{(Logarithms)}\\
&= \underbrace{\mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z))}_{\mathcal{L}(x^{(i)},\theta,\phi)\ \textbf{“Elbow”}} + \underbrace{D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z \mid x^{(i)}))}_{\geq 0}
\end{aligned}
$$

Credit: Fei-Fei Li, Andrej Karpathy and Justin Johnson, CS231n, Stanford Univ

Variational Autoencoder: The Math

$$\log p_\theta(x^{(i)}) = \mathbf{E}_{z\sim q_\phi(z|x^{(i)})}\left[\log p_\theta(x^{(i)})\right] \quad (p_\theta(x^{(i)}) \text{ Does not depend on } z)$$

$$= \mathbf{E}_z\left[\log \frac{p_\theta(x^{(i)} \mid z)p_\theta(z)}{p_\theta(z \mid x^{(i)})}\right] \quad \text{(Bayes' Rule)}$$

$$= \mathbf{E}_z\left[\log \frac{p_\theta(x^{(i)} \mid z)p_\theta(z)}{p_\theta(z \mid x^{(i)})}\frac{q_\phi(z \mid x^{(i)})}{q_\phi(z \mid x^{(i)})}\right] \quad \text{(Multiply by constant)}$$

$$= \mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - \mathbf{E}_z\left[\log \frac{q_\phi(z \mid x^{(i)})}{p_\theta(z)}\right] + \mathbf{E}_z\left[\log \frac{q_\phi(z \mid x^{(i)})}{p_\theta(z \mid x^{(i)})}\right] \quad \text{(Logarithms)}$$

$$= \underbrace{\mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z))}_{\mathcal{L}(x^{(i)}, \theta, \phi)} + \underbrace{D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z \mid x^{(i)}))}_{\geq 0}$$

"Elbow"

$$\log p_\theta(x^{(i)}) \geq \mathcal{L}(x^{(i)}, \theta, \phi)$$

Variational lower bound (elbow)

$$\theta^*, \phi^* = \arg\max_{\theta,\phi} \sum_{i=1}^{N} \mathcal{L}(x^{(i)}, \theta, \phi)$$

Training: Maximize lower bound

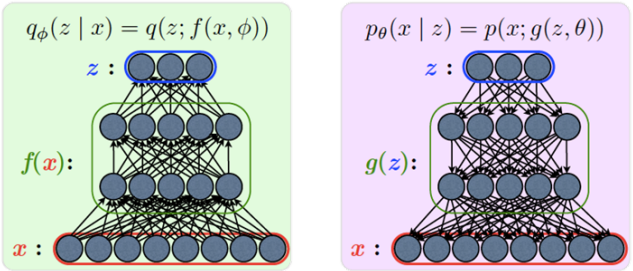
Credit: Fei-Fei Li, Andrej Karpathy and Justin Johnson, CS231n, Stanford Univ

Variational Autoencoder: Inference

- Introduce an inference model $q_\phi(z|x)$ that learns to approximates the intractable posterior $p_\theta(z|x)$ by optimizing the variational lower bound:

$$\mathcal{L}(\theta, \phi, x) = -D_{KL}(q_\phi(z|x)||p_\theta(z)) + \mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x|z)]$$

- We parametrize $q_\phi(z|x)$ with another neural network:



Credit: Aaron Courville, Deep Learning Summer School, 2015

Variational Autoencoder: How to train?

$$\begin{aligned}\mathcal{L}_{VAE} &= \mathbb{E}_{q_\phi(z|x)}\left[\log \frac{p_\theta(z,x)}{q_\phi(z|x)}\right] \\ &= -D_{KL}(q_\phi(z|x)||p_\theta(z)) + \mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x|z)]\end{aligned}$$

- $z \sim q_\phi(z|x)$: need to differentiate through the sampling process; how to update ϕ? (encoder is probabilistic)

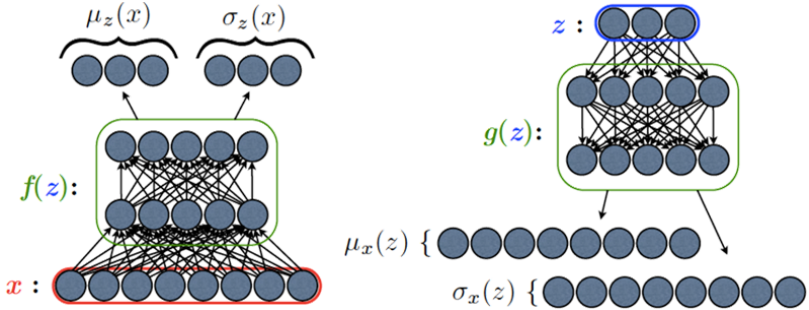
Variational Autoencoder: How to train?

$$\begin{aligned}\mathcal{L}_{VAE} &= \mathbb{E}_{q_\phi(z|x)}\left[\log \frac{p_\theta(z,x)}{q_\phi(z|x)}\right] \\ &= -D_{KL}(q_\phi(z|x)||p_\theta(z)) + \mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x|z)]\end{aligned}$$

- $z \sim q_\phi(z|x)$: need to differentiate through the sampling process; how to update ϕ? (encoder is probabilistic)
- **Solution**: Make the randomness independent of encoder output, thus making the encoder deterministic; how?

Reparametrization Trick

- Let's consider z to be real and $q_\phi(z|x) = \mathcal{N}(z; u_z(x), \sigma_z(x))$
- Parametrize z as $z = \mu_z(z) + \sigma_z(x)\epsilon_z$ where $\epsilon_z = \mathcal{N}(0, 1)$

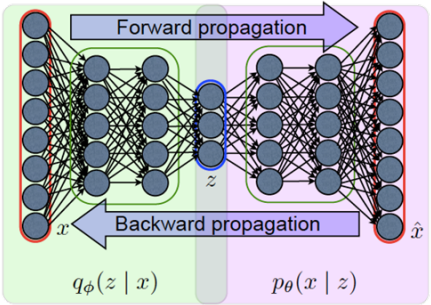


Credit: Aaron Courville, Deep Learning Summer School, 2015

Training with Backpropagation

With the **reparametrization trick**, we can simultaneously train both the **generative model** $p_\theta(x|z)$ and the **inference model** $q_\phi(z|x)$ using backpropagation

Objective function: $\mathcal{L}(\theta, \phi, x) = -D_{KL}(q_\phi(z|x)||p_\theta(z)) + \mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x|z)]$



Credit: Aaron Courville, Deep Learning Summer School, 2015

VAE: Summary

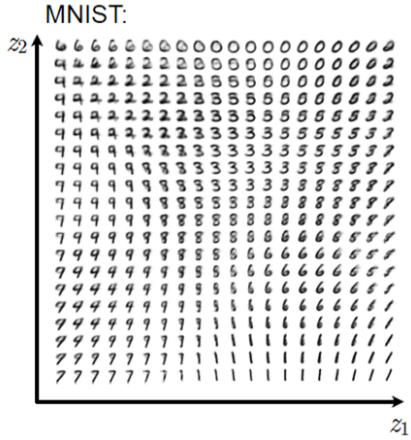
Traditional Autoencoders

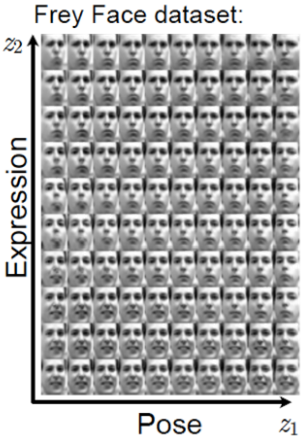
- Learned by reconstructing input
- Used to learn features, initialize supervised models (not much anymore though)

Variational Autoencoders

- Bayesian learning meets deep learning
- Sample from model to generate images

VAE: What can they do?

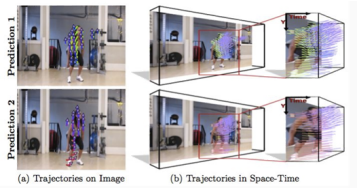


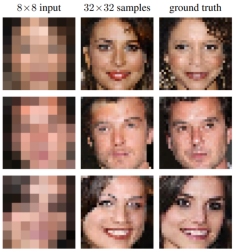


Credit: Aaron Courville, Deep Learning Summer School, 2015

Applications of VAEs

- Image and video generation
- Superresolution
- Forecasting from static images
- Image inpainting
- many more...





Credit: Dahl et al, Pixel Recursive Super Resolution, ICCV 2017

A Few Variants and Extensions

- Semi-Supervised VAEs
  - Kingma et al, Semi-Supervised Learning with Deep Generative Models, NeurIPS 2014
- Conditional VAE
  - Sohn et al, Learning Structured Output Representation using Deep Conditional Generative Models, NeurIPS 2015
- Importance-Weighted VAE
  - Burda et al, Importance Weighted Autoencoders, ICLR 2016
- Denoising VAE
  - Jiwoong et al, Denoising Criterion for Variational Auto-encoding Framework, AAAI 2017
- Inverse Graphics Network
  - Kulkarni et al, Deep Convolutional Inverse Graphics Network, NeurIPS 2015
- Adversarial Autoencoders
  - Makhzani et al, Adversarial Autoencoders, ICLR 2016

Homework

Readings

- Carl Doersch, Tutorial on Variational Autoencoder, arXiv 2016
- VAE example in PyTorch
- Kingma and Welling, Auto-Encoding Variational Bayes, ICLR 2014

Question

- Why does the encoder of a VAE map to a vector of means and a vector of standard deviations? Why does it not instead map to a vector of means and a covariance matrix?
- What about the decoder? If we assume a Mean Squared Error for the reconstruction loss, what is the covariance of the $p(x|z)$ Gaussian distribution?

Deep Learning for Computer Vision

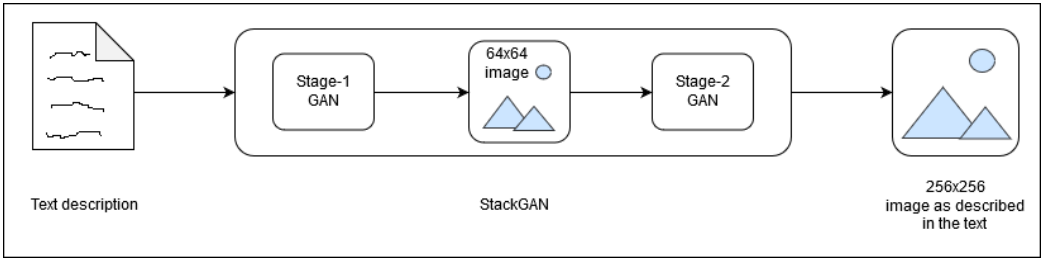
GAN Improvements

Vineeth N Balasubramanian

Department of Computer Science and Engineering
Indian Institute of Technology, Hyderabad

StackGAN[1]

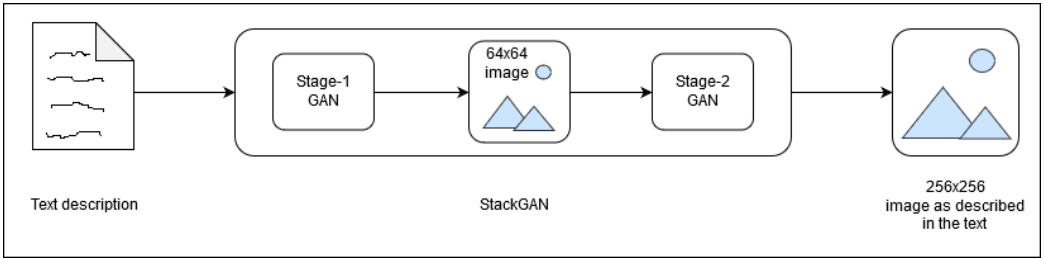
Generate 256 x 256 photo-realistic images conditioned on text descriptions



[1]Zhang et al, StackGAN: Text to Photo-realistic Image Synthesis with Stacked Generative Adversarial Networks, ICCV 2017

StackGAN[1]

Generate 256 x 256 photo-realistic images conditioned on text descriptions

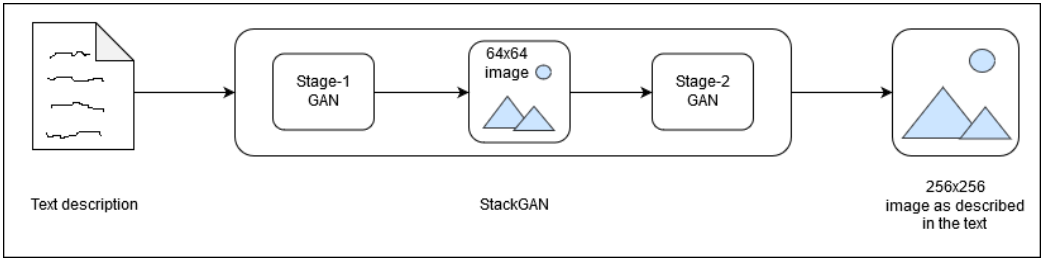


- **Stage 1:** Generate 64×64 images, low details

[1] Zhang et al, StackGAN: Text to Photo-realistic Image Synthesis with Stacked Generative Adversarial Networks, ICCV 2017

StackGAN[1]

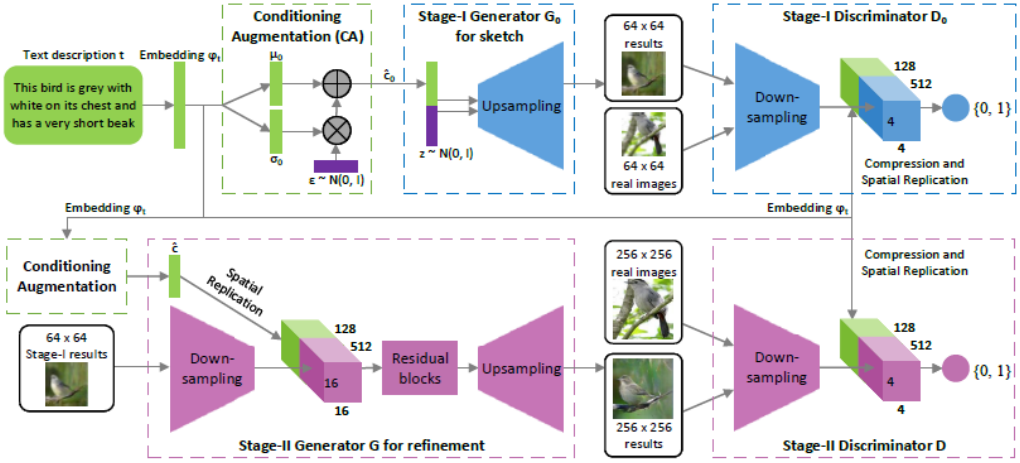
Generate 256 x 256 photo-realistic images conditioned on text descriptions



- **Stage 1:** Generate 64×64 images, low details
- **Stage 2:** Take Stage 1 output, generate 256×256, high detail and photo realistic, images
- Both stages conditioned on same textual input

[1] Zhang et al, StackGAN: Text to Photo-realistic Image Synthesis with Stacked Generative Adversarial Networks, ICCV 2017

StackGAN: Two-stage Network[2]



[2]Zhang et al, StackGAN: Text to Photo-realistic Image Synthesis with Stacked Generative Adversarial Networks, ICCV 2017

Loss Functions

Scores from Discriminator:

$$s_r \leftarrow D(x, h) \qquad \{\text{real image, correct text}\}$$
$$s_w \leftarrow D(x, \hat{h}) \qquad \{\text{real image, wrong text}\}$$
$$s_f \leftarrow D(\hat{x}, h) \qquad \{\text{fake image, correct text}\}$$

Loss Functions

Scores from Discriminator:

$$
\begin{aligned}
s_r &\leftarrow D(x, h) && \{\text{real image, correct text}\} \\
s_w &\leftarrow D(x, \hat{h}) && \{\text{real image, wrong text}\} \\
s_f &\leftarrow D(\hat{x}, h) && \{\text{fake image, correct text}\}
\end{aligned}
$$

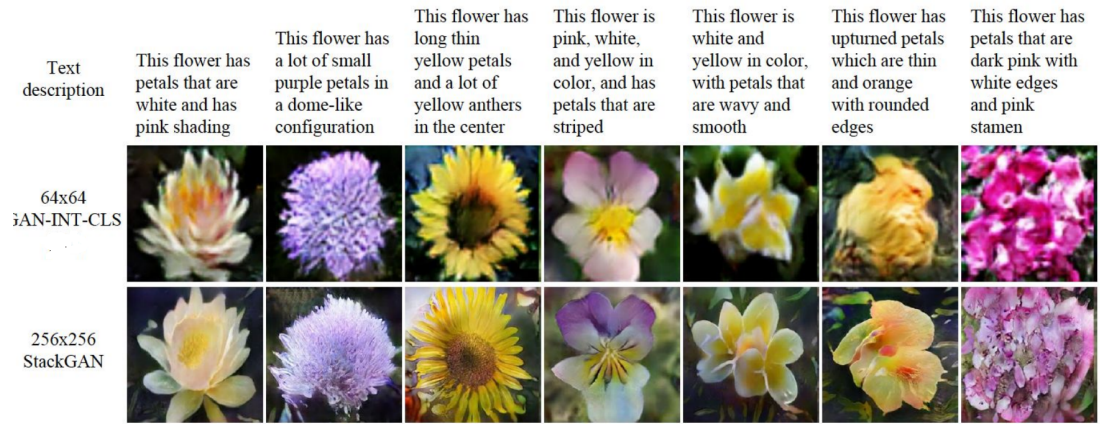
Then alternate maximizing:

$$\mathcal{L}_D \leftarrow \log(s_r) + (\log(1 - s_w) + \log(1 - s_f))/2$$

and minimizing:

$$\mathcal{L}_G \leftarrow \log(1 - s_f) + \lambda D_{KL}(\mathcal{N}(\mu_0(\phi_t), \Sigma_0(\phi_t))||\mathcal{N}(0, I))$$

StackGAN: Sample Results[3]



[3]Zhang et al, StackGAN: Text to Photo-realistic Image Synthesis with Stacked Generative Adversarial Networks, ICCV 2017; Reed et al, GAN-INT-CLS: Generative Adversarial Text to Image Synthesis, ICML 2016

Progressive GAN[4]

- Generates high-resolution images at 1024×1024 resolution
- **Key idea:** Grow both generator and discriminator progressively

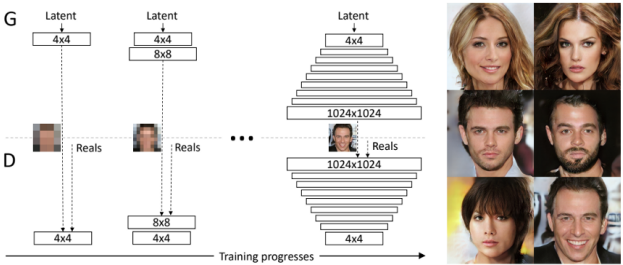
[4]Karras et al, Progressive Growing of GANs for Improved Quality, Stability, and Variation, ICLR 2018

Progressive GAN[4]

- Generates high-resolution images at 1024×1024 resolution
- **Key idea:** Grow both generator and discriminator progressively
- **Other contributions:** Minibatch standard deviation, equalized learning rate and pixel-wise feature vector normalization in generator

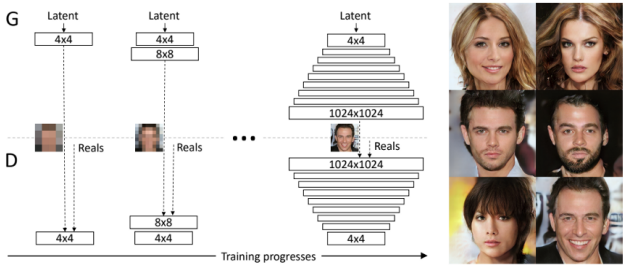
[4]Karras et al, Progressive Growing of GANs for Improved Quality, Stability, and Variation, ICLR 2018

Progressive GAN: Multi-scale Architecture



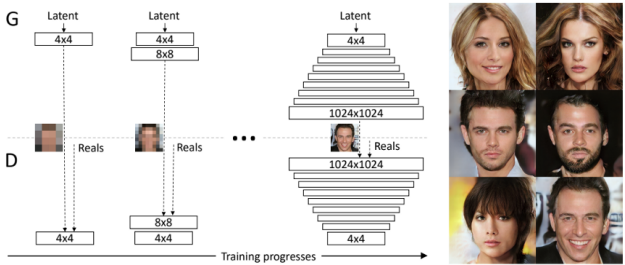
- Generator first produces 4×4 images until this reaches some kind of convergence

Progressive GAN: Multi-scale Architecture



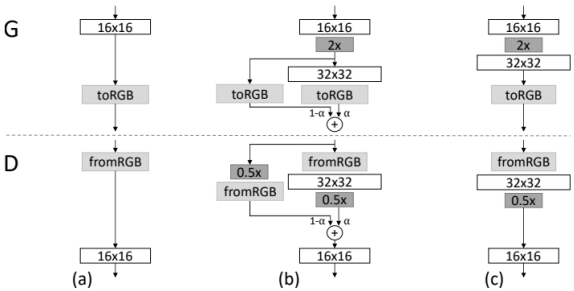
- Generator first produces 4×4 images until this reaches some kind of convergence
- Then task increases to 8×8 images, and so on until 1024×1024

Progressive GAN: Multi-scale Architecture



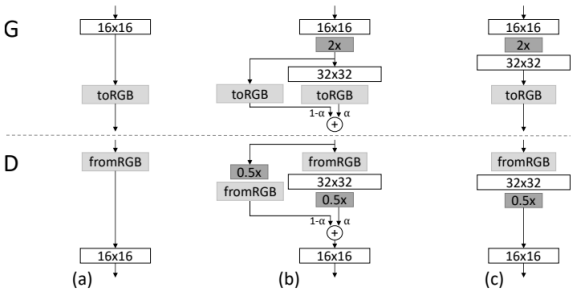
- Generator first produces 4×4 images until this reaches some kind of convergence
- Then task increases to 8×8 images, and so on until 1024×1024
- Allows for stable training of high-resolution images

Progressive GAN: Fading in New Layers



- Methodology similar to ResNets

Progressive GAN: Fading in New Layers



- Methodology similar to ResNets
- In figure (b), generator G:
  - **Nearest neighbor interpolation** of upsampled 16×16 layer's output, i.e. 32×32 is added to a 32×32 output layer
  - $\alpha \times$ new output layer $+ (1 - \alpha) \times$ projected layer; $\alpha \in \{0, 1\}$

Progressive GAN: Other Contributions

- **Minibatch standard deviation:** Standard deviation for each feature in each spatial location over a minibatch computed and averaged; this is concatenated to all spatial locations at a later layer of discriminator. Why?

Progressive GAN: Other Contributions

- **Minibatch standard deviation:** Standard deviation for each feature in each spatial location over a minibatch computed and averaged; this is concatenated to all spatial locations at a later layer of discriminator. Why? Homework!
- **Equalized learning rate:** $\tilde{w}_i = \frac{w_i}{c}$, where w_i are weights and c is per-layer normalization constant; helps keep weights at similar scale during training

Progressive GAN: Other Contributions

- **Minibatch standard deviation:** Standard deviation for each feature in each spatial location over a minibatch computed and averaged; this is concatenated to all spatial locations at a later layer of discriminator. Why? Homework!
- **Equalized learning rate:** $\tilde{w}_i = \frac{w_i}{c}$, where w_i are weights and c is per-layer normalization constant; helps keep weights at similar scale during training
- **Pixelwise feature vector normalization in generator G:** Normalize feature vector in each pixel of G after each convolutional layer using:

$$b_{x,y} = \frac{a_{x,y}}{\sqrt{\frac{1}{N}\sum_{j=0}^{N-1}(a_{x,y}^{j})^2 + \epsilon}} \tag{1}$$

where $\epsilon = 10^{-8}$, N is number of feature maps, $a_{x,y}$ and $b_{x,y}$ are original and normalized feature vectors in pixel (x, y) respectively

Progressive GAN: Results[5]

[5]Karras et al, Progressive Growing of GANs for Improved Quality, Stability, and Variation, ICLR 2018

StyleGAN[6]

- ProGAN generates high-quality images, but control of specific features is very limited

[6]Karras et al, A Style-Based Generator Architecture for Generative Adversarial Networks, CVPR 2019

StyleGAN[6]

- ProGAN generates high-quality images, but control of specific features is very limited
- **StyleGAN:** Automatically learned, unsupervised separation of high-level attributes (pose and identity), stochastic variation (hair) and scale-specific control attributes

[6]Karras et al, A Style-Based Generator Architecture for Generative Adversarial Networks, CVPR 2019

How StyleGAN works: Intuition

- **Coarse** resolution of up to 82 - affects pose, general hair style, face shape, etc

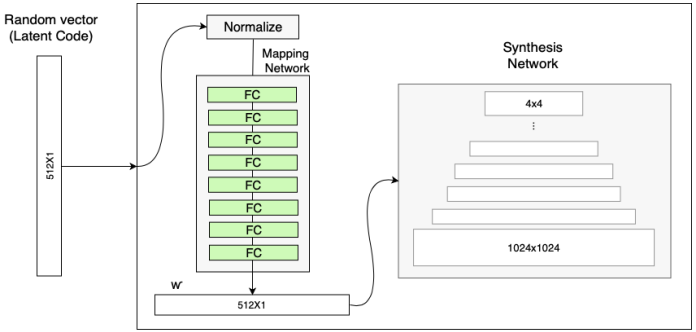
How StyleGAN works: Intuition

- **Coarse** resolution of up to 82 - affects pose, general hair style, face shape, etc
- **Middle** resolution of 162 to 322 - affects finer facial features, hair style, eyes open/closed, etc

How StyleGAN works: Intuition

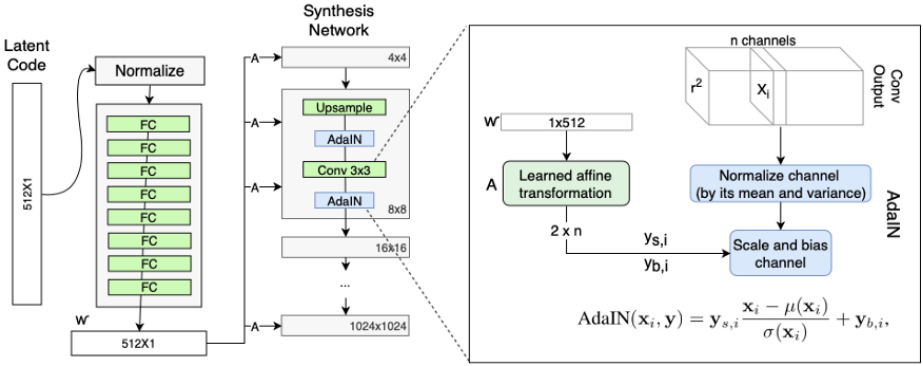
- **Coarse** resolution of up to 82 - affects pose, general hair style, face shape, etc
- **Middle** resolution of 162 to 322 - affects finer facial features, hair style, eyes open/closed, etc
- **Fine** resolution of 642 to 10242 - affects color scheme (eye, hair and skin) and micro features

StyleGAN: Mapping Network



- Encodes input vector into an intermediate vector to control different visual features
- 8 fully connected (FC) layers, output w is of same size as input layer, i.e. 512×1

StyleGAN: Adaptive Instance Normalization



- Transfers w (from mapping net) to generated image
- Module is added to each resolution level of synthesis network, defines the visual expression of features in that level

SPADE[7]

Key Idea

- Previous methods directly feed semantic layout as input to network

[7] Park et al, Semantic Image Synthesis with Spatially-Adaptive Normalization, CVPR 2019

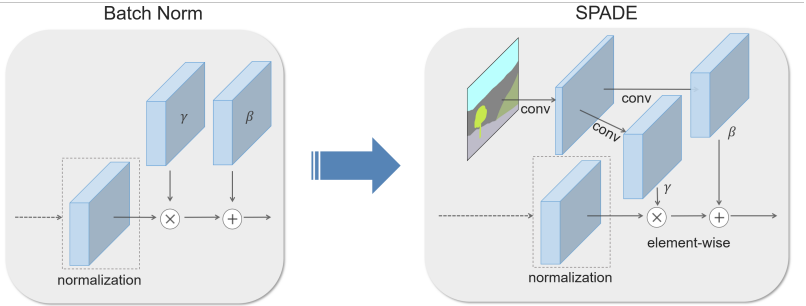
SPADE[7]

Key Idea

- Previous methods directly feed semantic layout as input to network
- **Spatially-adaptive normalization:** Input layout for modulating activations in normalization layers through a spatially-adaptive, learned transformation

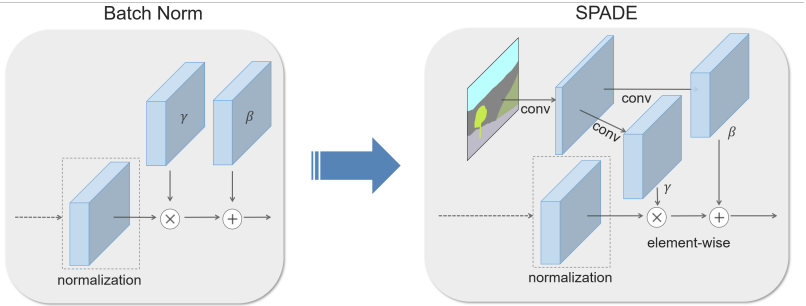
[7] Park et al, Semantic Image Synthesis with Spatially-Adaptive Normalization, CVPR 2019

SPADE: Methodology



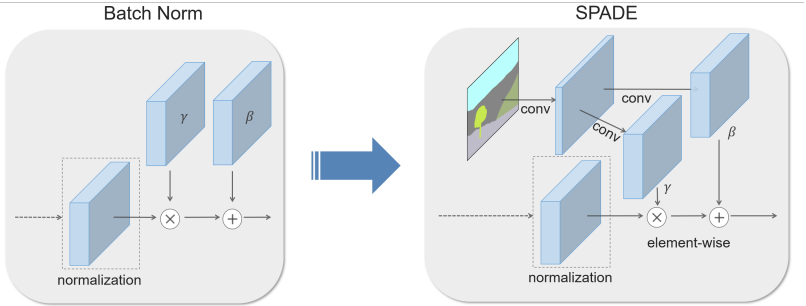
- Batch Normalization gives us affine layers

SPADE: Methodology



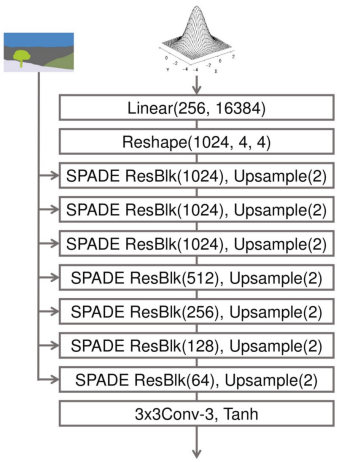
- Batch Normalization gives us affine layers
- In SPADE, affine layer is learned from semantic segmentation map (or any other computer vision task)

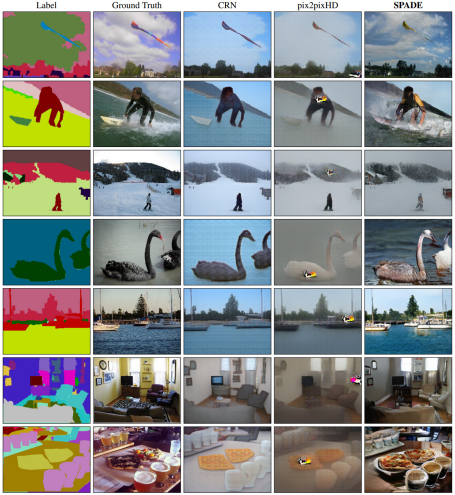
SPADE: Methodology



- Batch Normalization gives us affine layers
- In SPADE, affine layer is learned from semantic segmentation map (or any other computer vision task)
- Semantic information is provided via SPADE layers; random latent vector may still be used as input to network, used to manipulate style of generated images

SPADE: Architecture and Results[8]





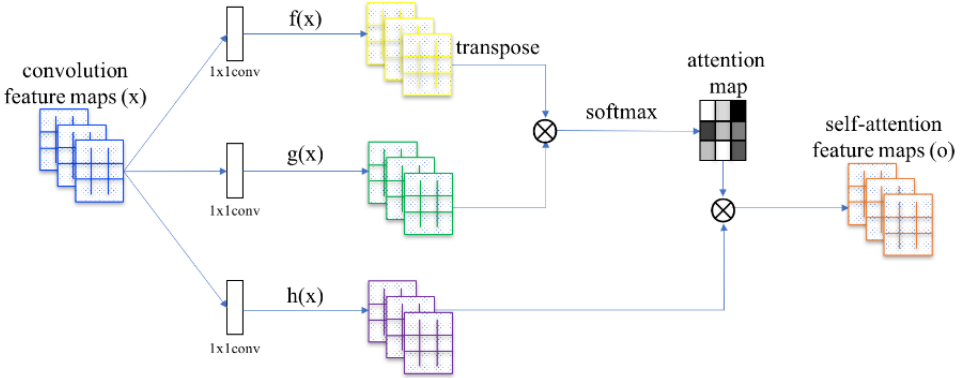
[8]Park et al, Semantic Image Synthesis with Spatially-Adaptive Normalization, CVPR 2019

BigGAN[9]

- Intended to scale up GANs for better high-resolution generation
- Designed for class-conditional image generation (generation of images using both a noise vector and class information as input)
- Multiple design decisions to improve generation quality

[9] Brock et al, Large Scale GAN Training for High Fidelity Natural Image Synthesis, ICLR 2019

BigGAN

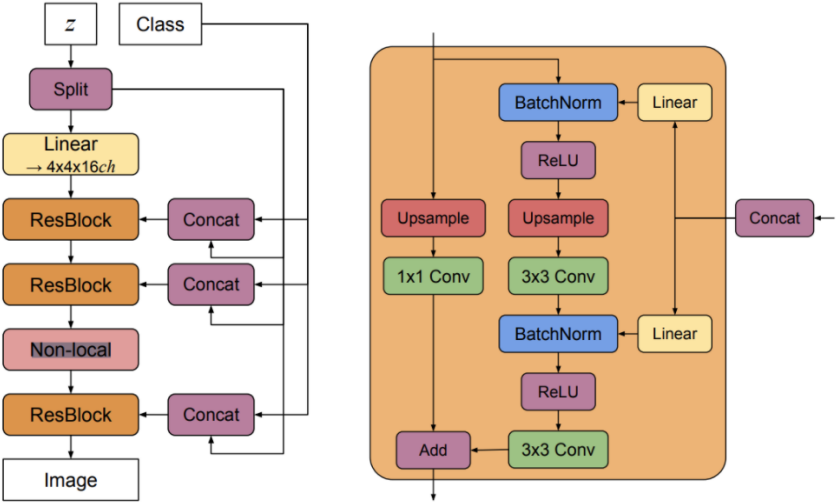


Base model is **Self-Attention GAN (SAGAN)**

Trained using **Hinge Loss:** $\max(0, 1 - t \cdot y)$, where t is target output and y is predicted output

Credit: Zhang et al, Self-Attention Generative Adversarial Networks, ICML 2019

BigGAN: Class-conditional Latents



BigGAN: Other Design Decisions

- **Spectral Normalization:** Normalizes weight matrix W using spectral norm so that it satisfies Lipschitz constraint $\sigma(W) = 1$ (See Miyato et al, Spectral Normalization for Generative Adversarial Networks, ICLR 2018 for details)
- **Orthogonal Weight Initialization:** Initialize weights in each layer to be a random orthogonal matrix (satisfying $W^T W = I$)
- **Skip-z Connections:** Directly connect input latent z to specific layers deep in the network
- **Orthogonal Regularization:** Encourages weights to be orthogonal: $R_\beta(W) = \beta ||W^T W - I||_F^2$. Why? Homework!

Credit: Jason Brownlee, MachineLearningMastery.com

BigGAN: Other Tricks/Hacks

- Updates discriminator model twice before updating generator model in each training iteration
- Model weights are averaged across prior training iterations using a moving average (similar to Progressive GAN)
- Large batch sizes of 256, 512, 1024 and 2048 images (best performance at 2048)
- More model parameters: doubled number of channels or feature maps (filters) in each layer
- **Truncation Trick:** Sample from truncated Gaussian (values above a threshold) as input at inference alone

Credit: Jason Brownlee, MachineLearningMastery.com

Homework

Readings

- Wang et al, Generative Adversarial Networks in Computer Vision: A Survey and Taxonomy, arXiv 2020
- Chapter 20 (Deep Generative Models), Deep Learning book
- Code links:
  - Progressive GAN: https://github.com/tkarras/progressive_growing_of_gans
  - StackGAN: https://github.com/hanzhanggit/StackGAN
  - StyleGAN: https://github.com/NVlabs/stylegan
  - BigGAN: https://github.com/ajbrock/BigGAN-PyTorch

Questions

- Minibatch Standard Deviation is used in ProgressiveGAN. Why is this useful?
- Orthogonal Regularization of weights is used in BigGAN. Why is this useful?

References

Tero Karras et al. "Progressive growing of gans for improved quality, stability, and variation". In: *arXiv preprint arXiv:1710.10196* (2017).

Han Zhang et al. "Stackgan: Text to photo-realistic image synthesis with stacked generative adversarial networks". In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 5907–5915.

Andrew Brock, Jeff Donahue, and Karen Simonyan. "Large Scale GAN Training for High Fidelity Natural Image Synthesis". In: *International Conference on Learning Representations*. 2018.

Tero Karras, Samuli Laine, and Timo Aila. "A style-based generator architecture for generative adversarial networks". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2019, pp. 4401–4410.

Taesung Park et al. "Semantic image synthesis with spatially-adaptive normalization". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 2337–2346.

Deep Learning for Computer Vision

Deep Generative Models: Image Applications

Vineeth N Balasubramanian

Department of Computer Science and Engineering
Indian Institute of Technology, Hyderabad

Homework

Question: Why is MI Gap and not MI used as a metric for disentanglement?

Homework

Question: Why is MI Gap and not MI used as a metric for disentanglement?

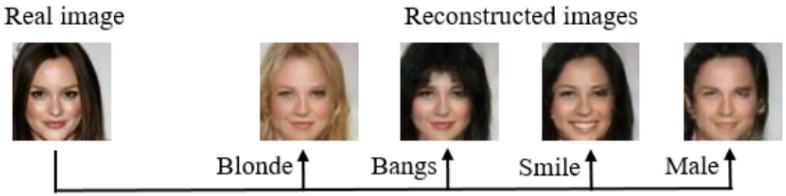
- MI Gap penalizes unaligned latent variables (contains information about more than one generative factor) $\rightarrow$ undesirable for disentanglement
- If one latent variable reliably models a generative factor, not required for other latent variables to be informative about this factor
- Read Chen et al, Isolating Sources of Disentanglement in Variational Autoencoders, NeurIPS 2018 for more information!

GANs for Image Editing

- Simple image edits such as grayscaling, brightness adjustment or contrast don't require an understanding of the image

GANs for Image Editing

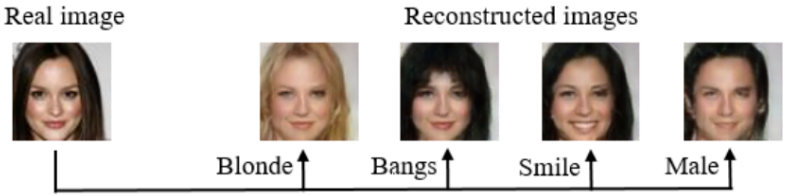
- Simple image edits such as grayscaling, brightness adjustment or contrast don't require an understanding of the image



- However, challenging operations such as changing attributes of a face require sound understanding of the input image. Can GANs help?

GANs for Image Editing

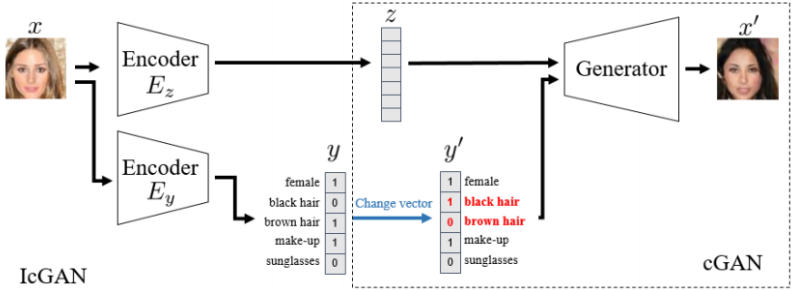
- Simple image edits such as grayscaling, brightness adjustment or contrast don't require an understanding of the image



- However, challenging operations such as changing attributes of a face require sound understanding of the input image. Can GANs help?
- But a GAN on its own does not have the mechanism to map a real image to its latent representation

Image Editing with Invertible Conditional GANs (IcGAN)[1]

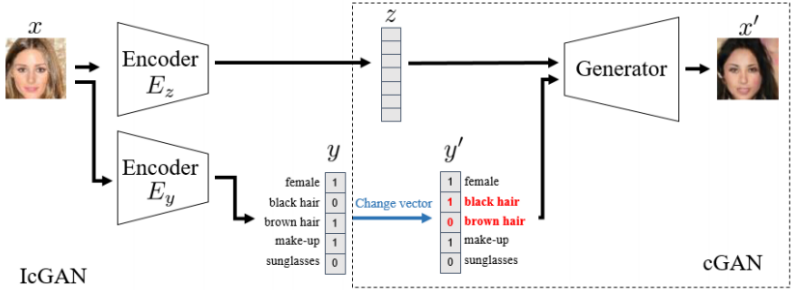
- Combines a conditional GAN with an encoder which can encode an image to its latent representation



[1] Perarnau et al, Invertible Conditional GANs for Image Editing, NeurIPS-W 2016

Image Editing with Invertible Conditional GANs (IcGAN)[1]

- Combines a conditional GAN with an encoder which can encode an image to its latent representation

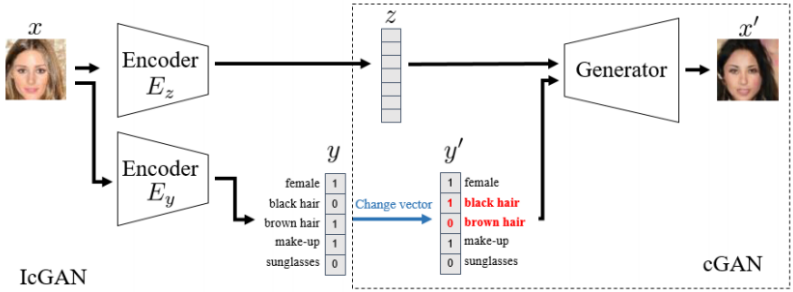


- Can be used for face editing as follows: (1) Pass image through encoder; (2) Change attribute vector y; (3) Pass through generator

[1] Perarnau et al, Invertible Conditional GANs for Image Editing, NeurIPS-W 2016

Invertible Conditional GANs

- Generator G samples an image x' from a latent representation z and conditional information y i.e., $G(z, y) = x'$



- Encoder performs the inverse i.e., $E(x) = (z, y)$

Training IcGAN

- First, a conditional GAN is trained to optimize the following objective:

$$\min_g \max_d v(\theta_g, \theta_d) = E_{x,y\sim p_{data}}[\log D(x,y)] + E_{z\sim p_z, x\sim p_x}[\log(1 - D(G(z,y'),y'))]$$

Training IcGAN

- First, a conditional GAN is trained to optimize the following objective:

$$\min_g \max_d v(\theta_g, \theta_d) = E_{x,y\sim p_{data}}[\log D(x,y)] + E_{z\sim p_z, x\sim p_x}[\log(1 - D(G(z,y'),y'))]$$

- Encoder has two parts: E_z which maps an image to its latent representation, and E_y which maps an image to its conditional information (attributes)

Training IcGAN

- First, a conditional GAN is trained to optimize the following objective:

$$\min_g \max_d v(\theta_g, \theta_d) = E_{x,y\sim p_{data}}[\log D(x,y)] + E_{z\sim p_z.x\sim p_x}[\log(1 - D(G(z,y'),y'))]$$

- Encoder has two parts: E_z which maps an image to its latent representation, and E_y which maps an image to its conditional information (attributes)
- To train E_z, generator is used to generate a dataset of images. Pairs of images and their latent representations (x', z) are used to train E_z using reconstruction loss:

$$\mathcal{L}_{ez} = E_{z\sim p_z.y'\sim p_y}\|z - E_z(G(z,y'))\|_2^2$$

Training IcGAN

- First, a conditional GAN is trained to optimize the following objective:

$$\min_g \max_d v(\theta_g, \theta_d) = E_{x,y\sim p_{data}}[\log D(x,y)] + E_{z\sim p_z.x\sim p_x}[\log(1 - D(G(z,y'),y'))]$$

- Encoder has two parts: E_z which maps an image to its latent representation, and E_y which maps an image to its conditional information (attributes)
- To train E_z, generator is used to generate a dataset of images. Pairs of images and their latent representations (x', z) are used to train E_z using reconstruction loss:

$$\mathcal{L}_{ez} = E_{z\sim p_z.y'\sim p_y}\|z - E_z(G(z,y'))\|_2^2$$

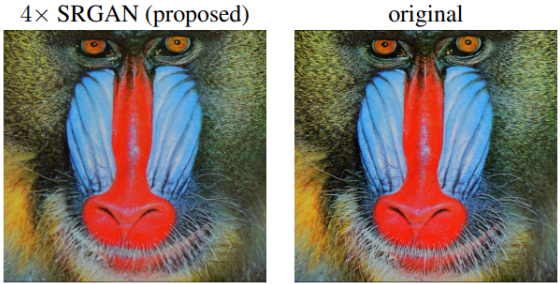
- E_y is trained using real image and attribute pairs:

$$\mathcal{L}_{ey} = E_{x,y\sim p_{data}}\|y - E_y(x)\|_2^2$$

IcGAN: Results on CelebA dataset

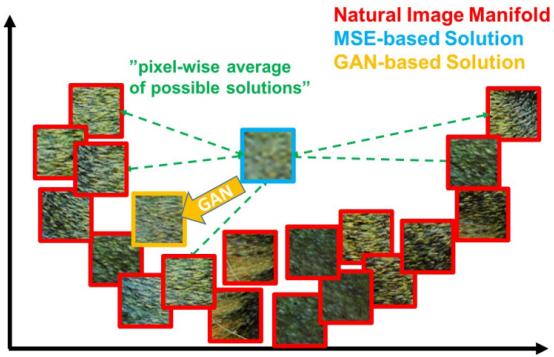
GANs for Super-Resolution[2]

- **Image Super-Resolution:** Task of obtaining a High-Resolution (HR) image from a Low Resolution (LR) image, a challenging task
- **SRGAN (Super-Resolution GAN)** aims to generate photo-realistic images with 4× upscaling factors



[2]Ledig et al, Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network, CVPR 2017

SR-GAN



- **Super-Resolution CNNs** minimize Mean Square Error (MSE) $\rightarrow$ tend to overly smoothen output images, due to pixel-wise averaging of outcomes
- **SR-GAN**, because of its adversarial objective, drives outputs to natural image manifold $\rightarrow$ results in high quality super-resolved images

SR-GAN: Content and Adversarial Losses

- Uses a pretrained VGG-19 network to compute **content loss** as MSE between VGG-19 feature representations of generator ($G_{\theta_G}(I^{LR})$) and true image I^{HR}:

$$l^{SR}_{VGG/i.j} = \frac{1}{W_{i,j}H_{i,j}} \sum_{x=1}^{W_{i,j}} \sum_{y=1}^{H_{i,j}} (\phi_{i,j}(I^{HR})_{x,y} - \phi_{i,j}(G_{\theta_G}(I^{LR}))_{x,y})^2$$

where $\phi_{i,j}$ = feature map obtained by j-th convolution before i-th max pooling layer, $W_{i,j}$ and $H_{i,j}$ are width and height of feature map respectively

SR-GAN: Content and Adversarial Losses

- Uses a pretrained VGG-19 network to compute **content loss** as MSE between VGG-19 feature representations of generator $(G_{\theta_G}(I^{LR}))$ and true image I^{HR}:

$$l^{SR}_{VGG/i.j} = \frac{1}{W_{i,j}H_{i,j}} \sum_{x=1}^{W_{i,j}} \sum_{y=1}^{H_{i,j}} (\phi_{i,j}(I^{HR})_{x,y} - \phi_{i,j}(G_{\theta_G}(I^{LR}))_{x,y})^2$$

  where $\phi_{i,j}$ = feature map obtained by j-th convolution before i-th max pooling layer, $W_{i,j}$ and $H_{i,j}$ are width and height of feature map respectively
- **Adversarial loss** given by:

$$l^{SR}_{Gen} = \sum_{n=1}^{N} -\log D_{\theta_D}(G_{\theta_G}(I^{LR}))$$

SR-GAN: Content and Adversarial Losses

- Uses a pretrained VGG-19 network to compute **content loss** as MSE between VGG-19 feature representations of generator ($G_{\theta_G}(I^{LR})$) and true image I^{HR}:

$$l^{SR}_{VGG/i.j} = \frac{1}{W_{i,j}H_{i,j}} \sum_{x=1}^{W_{i,j}} \sum_{y=1}^{H_{i,j}} (\phi_{i,j}(I^{HR})_{x,y} - \phi_{i,j}(G_{\theta_G}(I^{LR}))_{x,y})^2$$

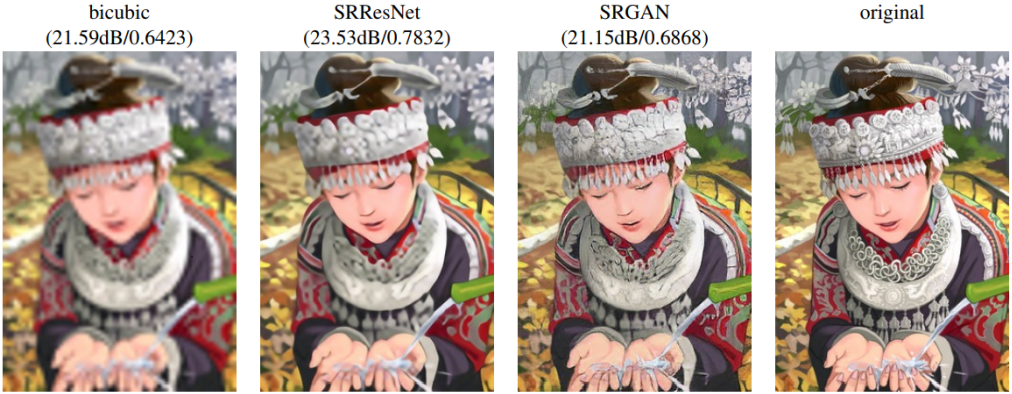
  where $\phi_{i,j}$ = feature map obtained by j-th convolution before i-th max pooling layer, $W_{i,j}$ and $H_{i,j}$ are width and height of feature map respectively
- **Adversarial loss** given by:

$$l^{SR}_{Gen} = \sum_{n=1}^{N} -\log D_{\theta_D}(G_{\theta_G}(I^{LR}))$$

- **Perceptual loss**, a weighted sum of content and adversarial loss:

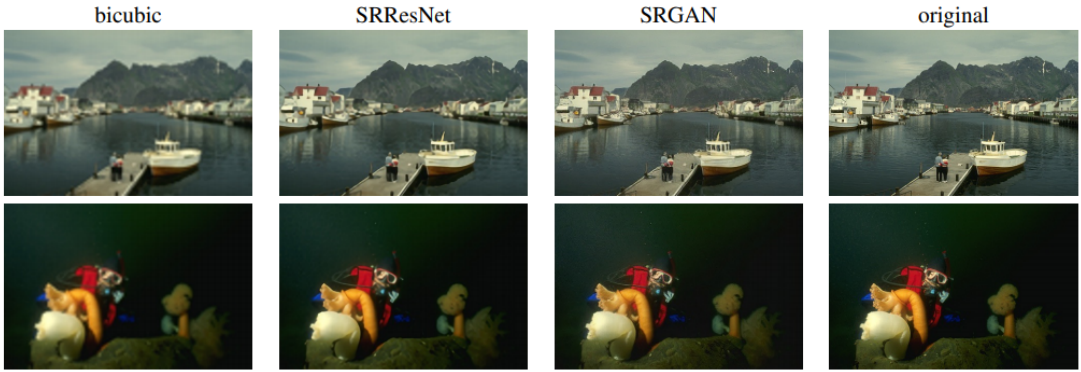
$$l^{SR} = l^{SR}_{\text{X}} + 10^{-3} l^{SR}_{Gen}$$

SR-GAN Results[3]



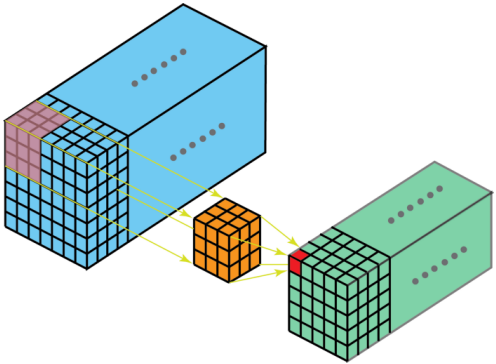
[3]Ledig et al, Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network, CVPR 2017

SR-GAN More Results[4]



[4]Ledig et al, Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network, CVPR 2017

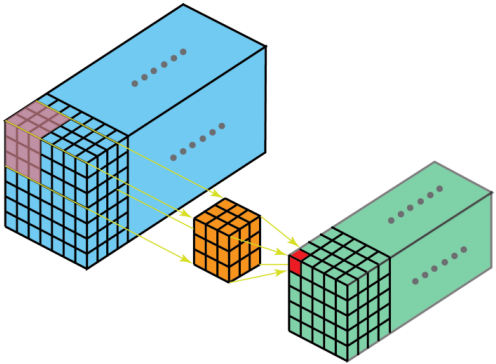
GANs for 3D Object Generation[5]



- We have seen GANs generate photo-realistic 2D images; can GANs generate 3D objects too?

[5] Wu et al, Learning a Probabilistic Latent Space of Object Shapes via 3D Generative Adversarial Modeling, NeurIPS 2016

GANs for 3D Object Generation[5]

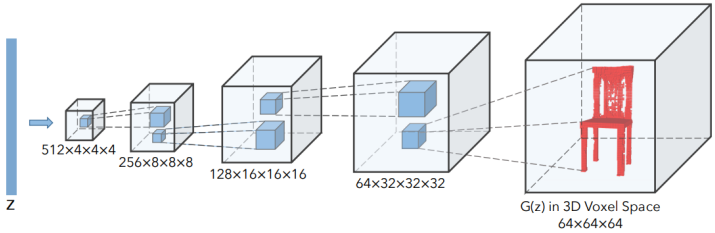


- We have seen GANs generate photo-realistic 2D images; can GANs generate 3D objects too?
- Yes! Recent advances in volumetric (3D) convolutional networks have made it possible to learn 3D object representations

[5] Wu et al, Learning a Probabilistic Latent Space of Object Shapes via 3D Generative Adversarial Modeling, NeurIPS 2016

GANs for 3D Object Generation[6]

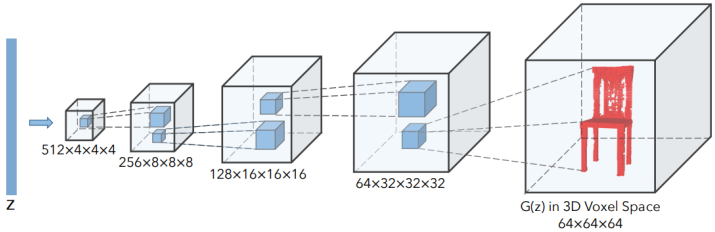
- Generator G maps a 200-length latent vector z to a $64 \times 64 \times 64$ cube, representing an object $G(z)$ in 3D-voxel space



[6]Wu et al, Learning a Probabilistic Latent Space of Object Shapes via 3D Generative Adversarial Modeling, NeurIPS 2016

GANs for 3D Object Generation[6]

- Generator G maps a 200-length latent vector z to a $64 \times 64 \times 64$ cube, representing an object $G(z)$ in 3D-voxel space

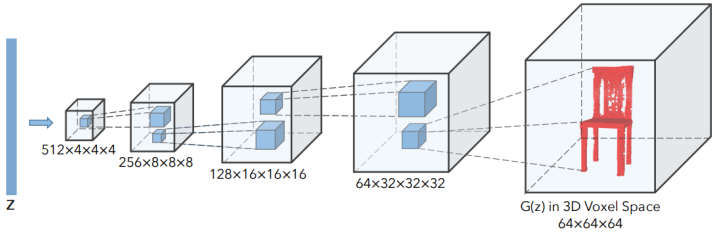


- Discriminator D classifies whether an input 3D object x is real or synthetic

[6]Wu et al, Learning a Probabilistic Latent Space of Object Shapes via 3D Generative Adversarial Modeling, NeurIPS 2016

GANs for 3D Object Generation[6]

- Generator G maps a 200-length latent vector z to a $64 \times 64 \times 64$ cube, representing an object $G(z)$ in 3D-voxel space

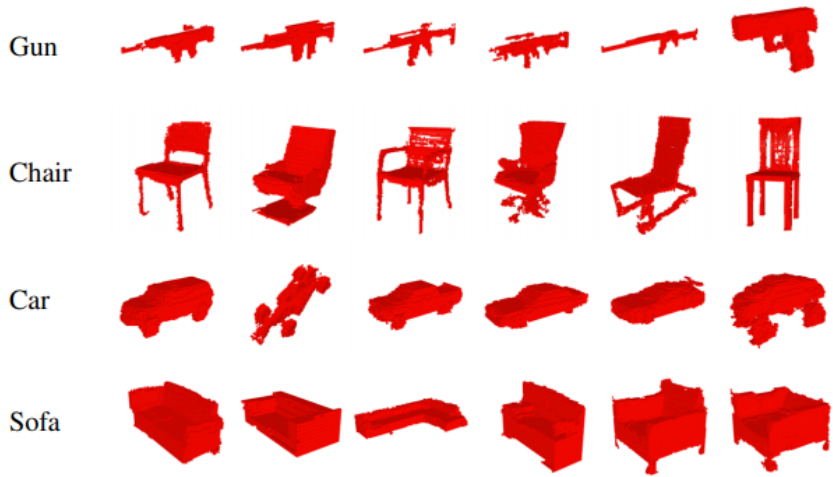


- Discriminator D classifies whether an input 3D object x is real or synthetic
- Adversarial loss in 3D-GAN:

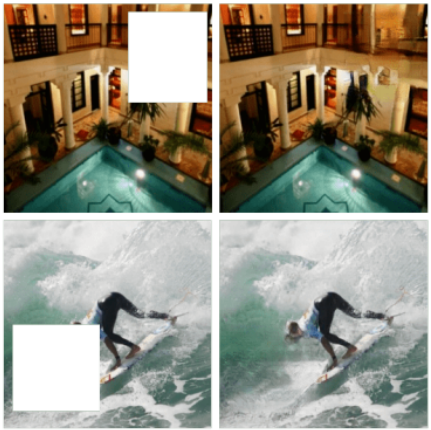
$$L_{3DGAN} = \log D(x) + \log(1 - D(G(z)))$$

[6]Wu et al, Learning a Probabilistic Latent Space of Object Shapes via 3D Generative Adversarial Modeling, NeurIPS 2016

3D-GAN Generated Objects



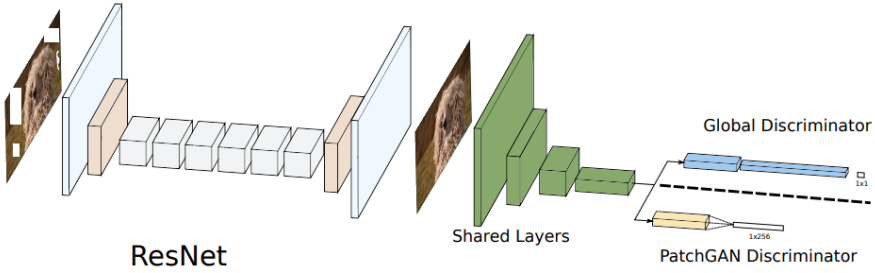
GANs for Image Inpainting[7]



- **Image inpainting** a reconstruction technique used for filling missing parts in an image
- Can a GAN be trained to perform image inpainting?

[7] Demir and Unal, Patch-Based Image Inpainting with Generative Adversarial Networks, arXiv 2018

PG-GAN[8]



- Consists of a ResNet-based generator and a novel discriminator with two heads:
  - **G-GAN discriminator:** reasons globally at image level (decide real vs fake)
  - **Patch GAN discriminator:** reasons locally at patch level (decide real vs fake) - helps capture local texture details

[8] Demir and Unal, Patch-Based Image Inpainting with Generative Adversarial Networks, arXiv 2018

PG-GAN Objective

- **Reconstruction loss:** pixel-wise L1 distance between generated image and ground truth:

$$\mathcal{L}_{rec} = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{WHC}\|y - x\|_1$$

- **Adversarial loss:** standard GAN objective:

$$\mathcal{L}_{GAN}(G, D) = E_{x\sim p(x)}[\log D(x)] + E_{y\sim p_G(x')}[\log(1 - D(G(x')))]$$

where $\mathcal{L}_{g\_adv}$ = adversarial loss for global GAN; $\mathcal{L}_{p\_adv}$ = adversarial loss for Patch GAN

- Overall objective:

$$\mathcal{L} = \lambda_1\mathcal{L}_{rec} + \lambda_2\mathcal{L}_{g\_adv} + \lambda_3\mathcal{L}_{p\_adv}$$

PG-GAN Results[9]

[9] Demir and Unal, Patch-Based Image Inpainting with Generative Adversarial Networks, arXiv 2018

PG-GAN More Results[10]

[10] Demir and Unal, Patch-Based Image Inpainting with Generative Adversarial Networks, arXiv 2018

Homework

Readings

- 3D-GANs
- Avinash H, GAN Zoo
- (Optional) Respective papers

References

Guim Perarnau et al. "Invertible Conditional GANs for image editing". In: *ArXiv* abs/1611.06355 (2016).

Jiajun Wu et al. "Learning a probabilistic latent space of object shapes via 3d generative-adversarial modeling". In: *Advances in Neural Information Processing Systems*. 2016, pp. 82–90.

C. Ledig et al. "Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network". In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017, pp. 105–114.

Ugur Demir and Gözde B. Ünal. "Patch-Based Image Inpainting with Generative Adversarial Networks". In: *CoRR* abs/1803.07422 (2018). arXiv: 1803.07422.