Generative Models

Deep Learning and AI, WS20/21

Dr. Yinchong Yang

Siemens AG

23.12.2020

Table of Contents

Table of Contents

Introduction

Task: generating data samples from a distribution.

Key question: how is the distribution represented?

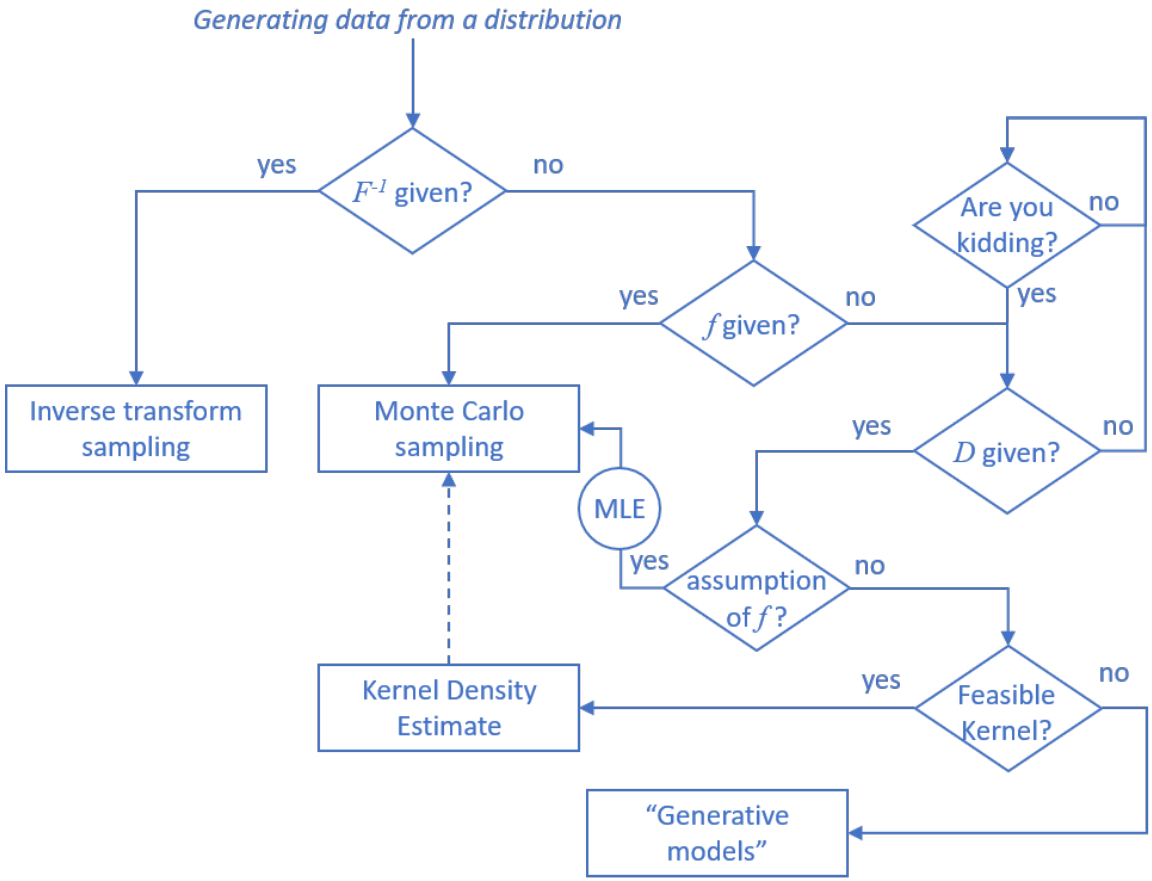


Figure: Landscape of some data generation methods.

Introduction

- Distribution represented by F^{-1}: e.g. Inverse transform sampling.

$$u \sim U(0,1), x = F^{-1}(u), x \sim p_X$$

- Distribution represented by f: e.g. Metropolis-Hastings sampling.

$$x^* \sim \mathcal{N}(x^{[t-1]}, \sigma),$$

$$x^{[t]} = x^* \quad \text{with probability} \quad \alpha = \frac{f(x^*)}{f(x^{[t-1]})}$$

Introduction

Distribution represented by data samples $\mathcal{D} = \{x_i\}_{i=1}^N$:

- Maximum likelihood estimate of parameters:

$$x \sim f(\widehat{\theta}_{MLE})$$

- Kernel density estimation:

$$\hat{f}(x) = \frac{1}{N} \sum_{i=1}^{N} K_h(x - x_i)$$

$$x \sim \mathcal{N}(x_i, h), \ i \sim U(0, N).$$

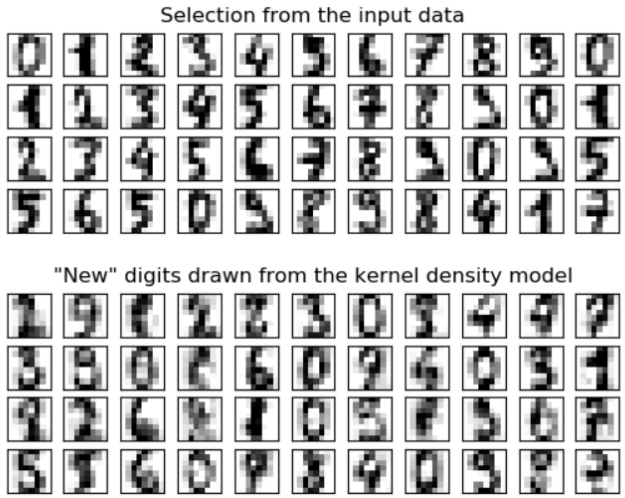


Figure: Example from `https://scikit-learn.org/stable/modules/density.html`, $p = 8 \times 8 = 64$, PCA(15), Gaussian Kernel with $h = 3.79$

Table of Contents

Autoencoder

Recap

A simple neural network architecture where input and target are identical.

- Given unlabeled data

$$\mathcal{D} = \{\boldsymbol{x}_i\}_{i=1}^N, \boldsymbol{x}_i \in \mathbb{R}^p.$$

- Encoder:

$$\boldsymbol{z} = E_\phi(\boldsymbol{x}) \in \mathbb{R}^k.$$

- Decoder:

$$\widehat{\boldsymbol{x}} = D_\theta(\boldsymbol{z}) \in \mathbb{R}^p.$$

- Objective:

$$\frac{1}{N}\sum_{i=1}^N ||\boldsymbol{x}_i - \widehat{\boldsymbol{x}}_i||_2^2$$

- where $k < p$

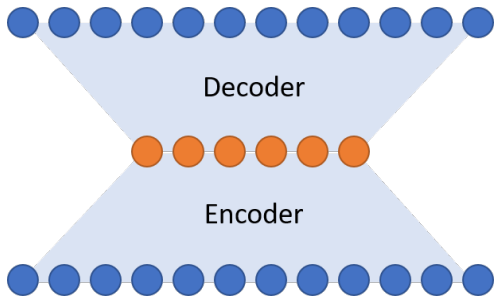


Figure: Autoencoder in its simplest form.

Autoencoder

Recap

- Pre-training of deep neural networks, with or without fine-tuning.
- As generic feature extractor / dimension reduction.
  - SVD has an AE-like interpretation: $\boldsymbol{X} \overset{\simeq}{=} \boldsymbol{U}_r \boldsymbol{D}_r \boldsymbol{V}_r^T = \boldsymbol{X} \boldsymbol{V}_r \boldsymbol{V}_r^T$
    - More flexible with arbitrary neural networks.

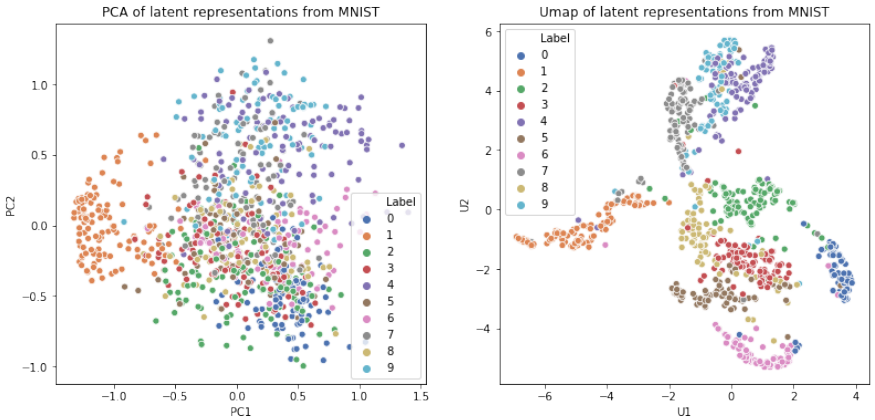


Figure: Illustrations of dimension reductions: AE+PCA(left) and AE+Umap(right)

- Challenges to sample from the latent space: i) discrete samples and ii) no distribution assumption.

Variational Autoencoder

Definition

Proposed by [Kingma and Welling 2013]

- Encoder:

$$\begin{bmatrix} \boldsymbol{\mu} \\ \boldsymbol{\sigma} \end{bmatrix} = E_\phi(\boldsymbol{x}), \ \boldsymbol{\mu}, \boldsymbol{\sigma} \in \mathbb{R}^k$$

$$\boldsymbol{z} \sim \mathcal{N}(\boldsymbol{\mu}, diag(\boldsymbol{\sigma}))$$

- Decoder:

$$\widehat{\boldsymbol{x}} = D_\theta(\boldsymbol{z})$$

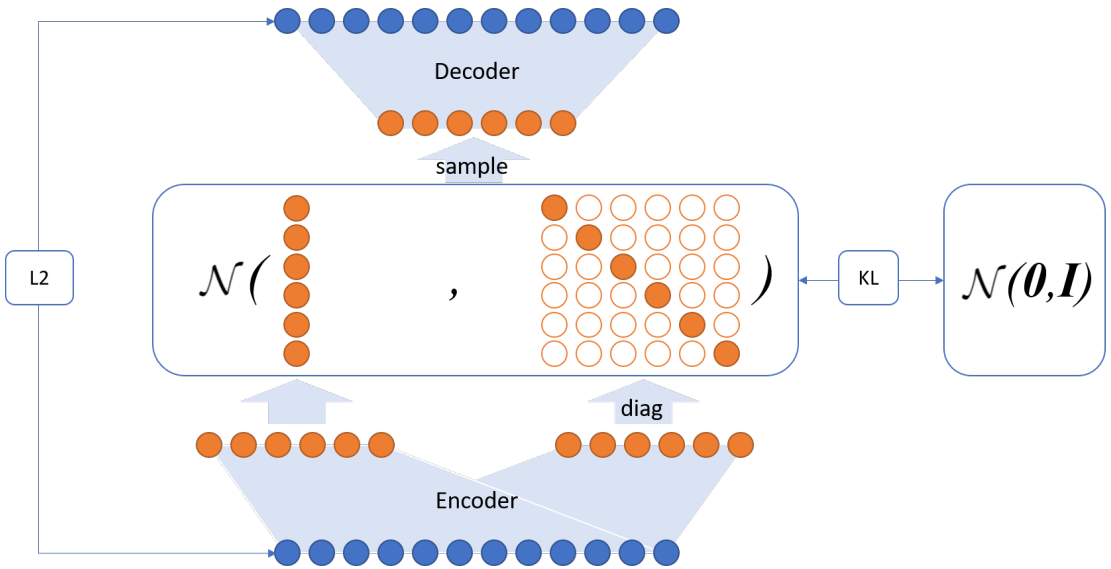


Figure: Illustration of a VAE architecture

Variational Autoencoder

Objectives

- Reconstruction loss: Similar to autoencoder, $\boldsymbol{z}$ should encode relevant information in $\boldsymbol{x}$

$$J^{rec} = ||\boldsymbol{x} - \widehat{\boldsymbol{x}}||_2^2$$

like an autoencoder, enforcing output close to input $\Rightarrow$ encoding relevant information in $\boldsymbol{z}$

- Latent loss: Additionally, in order to perform random sampling, $\mathcal{N}(\mu, diag(\boldsymbol{\sigma}))$ has to be similar to a known distribution.

$$J^{lat} = KL\left(\mathcal{N}(\boldsymbol{\mu}, diag(\boldsymbol{\sigma}))||\mathcal{N}(\boldsymbol{0}, \boldsymbol{I})\right)$$

enforcing the distribution of z to be close to a known distribution for inference.

Variational Autoencoder

Objectives

- The re-parametrization trick:

$$\boldsymbol{z} \sim \mathcal{N}(\boldsymbol{\mu}, diag(\boldsymbol{\sigma})) \Longleftrightarrow \boldsymbol{z} = \boldsymbol{\mu} + \boldsymbol{\epsilon} \circ \boldsymbol{\sigma}, \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$$
$$\Rightarrow \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{u}} = \mathbf{1}, \ \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{\sigma}} = \boldsymbol{\epsilon}$$

- KL divergence to measure the distance between two distributions

$$KL(P||Q) = \int_x p(x) \log \left(\frac{p(x)}{q(x)} \right) dx$$

For two Gaussian specifically:

$$KL\left(\mathcal{N}(\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1)||\mathcal{N}(\boldsymbol{\mu}_2, \boldsymbol{\Sigma}_2)\right)$$
$$= \frac{1}{2} \left(tr(\boldsymbol{\Sigma}_2^{-1}\boldsymbol{\Sigma}_1) + (\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1)^T \boldsymbol{\Sigma}_2^{-1}(\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1) - k + \ln \frac{|\boldsymbol{\Sigma}_2|}{|\boldsymbol{\Sigma}_1|} \right)$$

Variational Autoencoder

Implementation in TF

```
import tensorflow_probability as tfp
# Input: ##########################################
X = tf.placeholder(tf.float32, [None, p])

# Encoder: ########################################
enc_h0 = tf.keras.layers.Dense(
    h_sizes[0],
    tf.nn.relu
)(X)
enc_h1 = tf.keras.layers.Dense(
    h_sizes[1],
    tf.nn.relu
)(enc_h0)
enc_loc = tf.keras.layers.Dense(Z_size)(enc_h1)
enc_scale = tf.keras.layers.Dense(
    Z_size,
    activation='softplus',
)(enc_h1)

# Distributions ###################################
enc_distr = tfp.distributions.MultivariateNormalDiag(
    enc_loc,
    enc_scale,
)
prior_distr = tfp.distributions.MultivariateNormalDiag(
    tf.zeros(Z_size),
    tf.ones(Z_size),
)
if True:  # automatic re-parametrization
    Z_sample = enc_distr.sample()
else:  # manual re-parametrization
    Z_sample = prior_distr.sample()*enc_scale+enc_loc
```

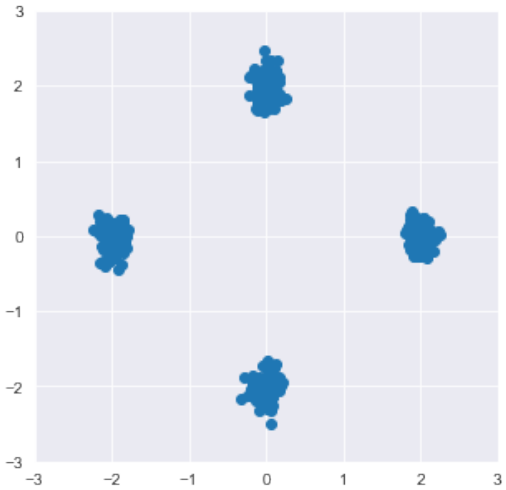
```
# Decoder #########################################
dec_h0 = tf.keras.layers.Dense(
    h_sizes[1],
    tf.nn.relu,
)(Z_sample)
dec_h1 = tf.keras.layers.Dense(
    h_sizes[0],
    tf.nn.relu,
)(dec_h0)
dec_X_out = tf.keras.layers.Dense(p)(dec_h1)

# Loss functions ##################################
recon_loss = tf.reduce_sum(
    tf.squared_difference(dec_X_out, X),
    1,
)
latent_loss = tf.distributions.kl_divergence(
    enc_distr,
    prior_distr,
)
loss = tf.reduce_mean(recon_loss + latent_loss)
```

Variational Autoencoder

Demo

- Data: $N = 100, p = 32$
- Manifold: four modes in 2-D space



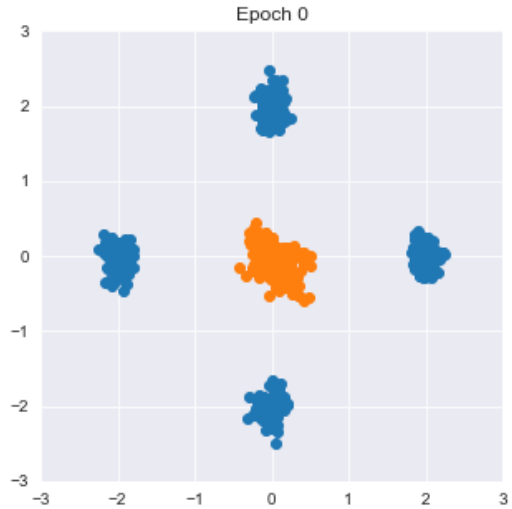


Figure: Distribution on the manifold.

Variational Autoencoder

Theory

The interpretation of the objective functions:

$$\log p(x) = \log \int_z p(x,z)dz \quad | \text{ Law of total prob.}$$

$$= \log \int_z p(x,z)\frac{q(z|x)}{q(z|x)}dz = \log \int_z q(z|x)\frac{p(x,z)}{q(z|x)}dz$$

$$= \log \mathbb{E}_{z\sim q(z|x)}\left[\frac{p(x,z)}{q(z|x)}\right] = \log \mathbb{E}_{z\sim q(z|x)}\left[\frac{p(x|z)p(z)}{q(z|x)}\right] \quad | \text{ Def. } \mathbb{E}$$

$$\geq \mathbb{E}_{z\sim q(z|x)}\left[\log \frac{p(x|z)p(z)}{q(z|x)}\right] \quad | \text{ Jensen Ineq.}$$

$$= \mathbb{E}_{z\sim q(z|x)}\left[\log p(x|z)\right] + \mathbb{E}_{z\sim q(z|x)}\left[-\log \frac{q(z|x)}{p(z)}\right]$$

$$= \mathbb{E}_{z\sim q(z|x)}\left[\log p(x|z)\right] - \int_z q(z|x)\cdot \log \frac{q(z|x)}{p(z)}dz \quad | \text{ Def. } \mathbb{E}$$

$$= \mathbb{E}_{z\sim q(z|x)}\left[\log p(x|z)\right] - KL(q(z|x)||p(z)) \quad | \text{ Def. } KL$$

Variational Autoencoder

Theory

The interpretation of the objective functions:

$$\log p(x) = \log \int_z p(x,z)dz \quad | \text{ Law of total prob.}$$

$$= \log \int_z p(x,z)\frac{q(z|x)}{q(z|x)}dz = \log \int_z q(z|x)\frac{p(x,z)}{q(z|x)}dz$$

$$= \log \mathbb{E}_{z\sim q(z|x)}\left[\frac{p(x,z)}{q(z|x)}\right] = \log \mathbb{E}_{z\sim q(z|x)}\left[\frac{p(x|z)p(z)}{q(z|x)}\right] \quad | \text{ Def. } \mathbb{E}$$

$$\geq \mathbb{E}_{z\sim q(z|x)}\left[\log \frac{p(x|z)p(z)}{q(z|x)}\right] \quad | \text{ Jensen Ineq.}$$

$$= \mathbb{E}_{z\sim q(z|x)}\left[\log p(x|z)\right] + \mathbb{E}_{z\sim q(z|x)}\left[-\log \frac{q(z|x)}{p(z)}\right]$$

$$= \mathbb{E}_{z\sim q(z|x)}\left[\log p(x|z)\right] - \int_z q(z|x)\cdot \log \frac{q(z|x)}{p(z)}dz \quad | \text{ Def. } \mathbb{E}$$

$$= \mathbb{E}_{z\sim q(z|x)}\left[\log p(x|z)\right] - KL(q(z|x)||p(z)) \quad | \text{ Def. } KL$$

Variational Autoencoder

Theory

The interpretation of the objective functions:

$$
\begin{aligned}
\log p(x) &= \log \int_z p(x,z) dz && \text{| Law of total prob.} \\
&= \log \int_z p(x,z) \frac{q(z|x)}{q(z|x)} dz = \log \int_z q(z|x) \frac{p(x,z)}{q(z|x)} dz \\
&= \log \mathbb{E}_{z\sim q(z|x)} \left[\frac{p(x,z)}{q(z|x)} \right] = \log \mathbb{E}_{z\sim q(z|x)} \left[\frac{p(x|z)p(z)}{q(z|x)} \right] && \text{| Def. } \mathbb{E} \\
&\geq \mathbb{E}_{z\sim q(z|x)} \left[\log \frac{p(x|z)p(z)}{q(z|x)} \right] && \text{| Jensen Ineq.} \\
&= \mathbb{E}_{z\sim q(z|x)} \left[\log p(x|z) \right] + \mathbb{E}_{z\sim q(z|x)} \left[-\log \frac{q(z|x)}{p(z)} \right] \\
&= \mathbb{E}_{z\sim q(z|x)} \left[\log p(x|z) \right] - \int_z q(z|x) \cdot \log \frac{q(z|x)}{p(z)} dz && \text{| Def. } \mathbb{E} \\
&= \mathbb{E}_{z\sim q(z|x)} \left[\log p(x|z) \right] - KL(q(z|x)||p(z)) && \text{| Def. } KL
\end{aligned}
$$

Variational Autoencoder

Theory

The interpretation of the objective functions:

$$
\begin{aligned}
\log p(x) &= \log \int_z p(x,z)dz && \text{| Law of total prob.} \\
&= \log \int_z p(x,z)\frac{q(z|x)}{q(z|x)}dz = \log \int_z q(z|x)\frac{p(x,z)}{q(z|x)}dz \\
&= \log \mathbb{E}_{z\sim q(z|x)}\left[\frac{p(x,z)}{q(z|x)}\right] = \log \mathbb{E}_{z\sim q(z|x)}\left[\frac{p(x|z)p(z)}{q(z|x)}\right] && \text{| Def. } \mathbb{E} \\
&\geq \mathbb{E}_{z\sim q(z|x)}\left[\log \frac{p(x|z)p(z)}{q(z|x)}\right] && \text{| Jensen Ineq.} \\
&= \mathbb{E}_{z\sim q(z|x)}\left[\log p(x|z)\right] + \mathbb{E}_{z\sim q(z|x)}\left[-\log \frac{q(z|x)}{p(z)}\right] \\
&= \mathbb{E}_{z\sim q(z|x)}\left[\log p(x|z)\right] - \int_z q(z|x)\cdot \log \frac{q(z|x)}{p(z)}dz && \text{| Def. } \mathbb{E} \\
&= \mathbb{E}_{z\sim q(z|x)}\left[\log p(x|z)\right] - KL(q(z|x)||p(z)) && \text{| Def. } KL
\end{aligned}
$$

Variational Autoencoder

Theory

The interpretation of the objective functions:

$$
\begin{aligned}
\log p(x) &= \log \int_z p(x,z)dz && \text{| Law of total prob.} \\
&= \log \int_z p(x,z)\frac{q(z|x)}{q(z|x)}dz = \log \int_z q(z|x)\frac{p(x,z)}{q(z|x)}dz \\
&= \log \mathbb{E}_{z\sim q(z|x)}\left[\frac{p(x,z)}{q(z|x)}\right] = \log \mathbb{E}_{z\sim q(z|x)}\left[\frac{p(x|z)p(z)}{q(z|x)}\right] && \text{| Def. } \mathbb{E} \\
&\geq \mathbb{E}_{z\sim q(z|x)}\left[\log \frac{p(x|z)p(z)}{q(z|x)}\right] && \text{| Jensen Ineq.} \\
&= \mathbb{E}_{z\sim q(z|x)}\left[\log p(x|z)\right] + \mathbb{E}_{z\sim q(z|x)}\left[-\log \frac{q(z|x)}{p(z)}\right] \\
&= \mathbb{E}_{z\sim q(z|x)}\left[\log p(x|z)\right] - \int_z q(z|x)\cdot \log \frac{q(z|x)}{p(z)}dz && \text{| Def. } \mathbb{E} \\
&= \mathbb{E}_{z\sim q(z|x)}\left[\log p(x|z)\right] - KL(q(z|x)||p(z)) && \text{| Def. } KL
\end{aligned}
$$

Variational Autoencoder

Theory

The interpretation of the objective functions:

$$
\begin{aligned}
\log p(x) &= \log \int_z p(x,z) dz && \text{| Law of total prob.} \\
&= \log \int_z p(x,z) \frac{q(z|x)}{q(z|x)} dz = \log \int_z q(z|x) \frac{p(x,z)}{q(z|x)} dz \\
&= \log \mathbb{E}_{z\sim q(z|x)} \left[\frac{p(x,z)}{q(z|x)} \right] = \log \mathbb{E}_{z\sim q(z|x)} \left[\frac{p(x|z)p(z)}{q(z|x)} \right] && \text{| Def. } \mathbb{E} \\
&\geq \mathbb{E}_{z\sim q(z|x)} \left[\log \frac{p(x|z)p(z)}{q(z|x)} \right] && \text{| Jensen Ineq.} \\
&= \mathbb{E}_{z\sim q(z|x)} \left[\log p(x|z) \right] + \mathbb{E}_{z\sim q(z|x)} \left[-\log \frac{q(z|x)}{p(z)} \right] \\
&= \mathbb{E}_{z\sim q(z|x)} \left[\log p(x|z) \right] - \int_z q(z|x) \cdot \log \frac{q(z|x)}{p(z)} dz && \text{| Def. } \mathbb{E} \\
&= \mathbb{E}_{z\sim q(z|x)} \left[\log p(x|z) \right] - KL(q(z|x)||p(z)) && \text{| Def. } KL
\end{aligned}
$$

Variational Autoencoder

Theory

The interpretation of the objective functions:

$$
\begin{aligned}
\log p(x) &= \log \int_z p(x,z)dz && \text{| Law of total prob.} \\
&= \log \int_z p(x,z)\frac{q(z|x)}{q(z|x)}dz = \log \int_z q(z|x)\frac{p(x,z)}{q(z|x)}dz \\
&= \log \mathbb{E}_{z\sim q(z|x)}\left[\frac{p(x,z)}{q(z|x)}\right] = \log \mathbb{E}_{z\sim q(z|x)}\left[\frac{p(x|z)p(z)}{q(z|x)}\right] && \text{| Def. } \mathbb{E} \\
&\geq \mathbb{E}_{z\sim q(z|x)}\left[\log \frac{p(x|z)p(z)}{q(z|x)}\right] && \text{| Jensen Ineq.} \\
&= \mathbb{E}_{z\sim q(z|x)}\left[\log p(x|z)\right] + \mathbb{E}_{z\sim q(z|x)}\left[-\log \frac{q(z|x)}{p(z)}\right] \\
&= \mathbb{E}_{z\sim q(z|x)}\left[\log p(x|z)\right] - \int_z q(z|x)\cdot \log \frac{q(z|x)}{p(z)}dz && \text{| Def. } \mathbb{E} \\
&= \mathbb{E}_{z\sim q(z|x)}\left[\log p(x|z)\right] - KL(q(z|x)||p(z)) && \text{| Def. } KL
\end{aligned}
$$

Variational Autoencoder

Theory

What is $\mathbb{E}_{z\sim q(z|x)}[\log p(x|z)] - KL(q(z|x)||p(z))$?
Assuming p and q to be Gaussians:

- $\mathbb{E}_{z\sim q(z|x)}[\log p(x|z)]$ represents the reconstruction error:

$$\frac{1}{n}\sum_{i=1}^{n}\log p(x_i|z_i) = \frac{1}{n}\sum_{i=1}^{n}\log \mathcal{N}(x_i;\mu_i^D,\sigma_i^D),$$

$$\text{with } \mu_i^D = D(z_i), z_i \sim \mathcal{N}(\mu_i^E, diag(\sigma_i^E)), \begin{bmatrix}\mu_i^E \\ \sigma_i^E\end{bmatrix} = E(x_i)$$

- $-KL(q(z|x)||p(z))$ represents the latent loss:

$$q(z_i|x_i) = \mathcal{N}(z_i;\mu_i^E, diag(\sigma_i^E)), \begin{bmatrix}\mu_i^E \\ \sigma_i^E\end{bmatrix} = E(x_i),$$

$$p(z_i) = \mathcal{N}(z_i;0,1)$$

$\Rightarrow$ our objective function is the lower bound of the data likelihood $p(x)$.

Table of Contents

GAN

Definition

Proposed in [Goodfellow et al. 2014], earlier related work in [Schmidhuber 1992].

- Unlabeled data samples $\mathcal{D} = \{\boldsymbol{x}_i\}_{i=1}^N$, often denoted using epdf $p_X(\boldsymbol{x})$.
- A known distribution, usually uniform or MVG denoted with pdf $p_Z(\boldsymbol{z})$.
- A generator network $G(\boldsymbol{z})$.
- A discriminator network $D(\boldsymbol{x})$.

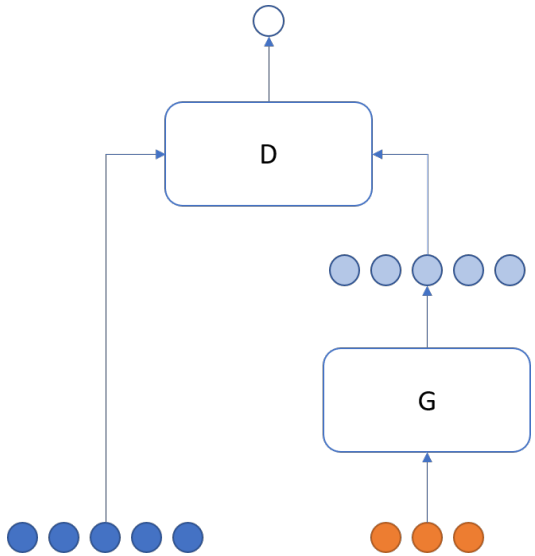


Figure: GAN architecture

GAN

Objectives

Recall in a supervised learning setting, the binary cross entropy(log likelihood of Bernoulli) of ground truth y and predicted probability based on input feature $\pi(\boldsymbol{x})$:

$$h(y, \pi) = y \cdot \log(\pi) + (1 - y) \cdot \log(1 - \pi) \tag{1}$$

- Discriminator's objective:
  Classifying real samples as real:

$$J_D^{real}(\theta_D) = h(1, D(\boldsymbol{x})) \overset{(1)}{=} \log(D(\boldsymbol{x}))$$

  Classifying generated samples as generated:

$$J_D^{gen}(\theta_D) = h(0, D(G(\boldsymbol{z}))) \overset{(1)}{=} \log(1 - D(G(\boldsymbol{z})))$$

  In total:

$$J_D(\theta_D) = J_D^{real}(\theta_D) + J_D^{gen}(\theta_D) = \log(D(\boldsymbol{x})) + \log(1 - D(G(\boldsymbol{z}))) \longrightarrow \max$$

- Generator's objective:

$$J_G(\theta_G) = h(1, D(G(\boldsymbol{z}))) \overset{(1)}{=} \log(D(G(\boldsymbol{z}))) \longrightarrow \max$$
$$\iff \log(1 - D(G(\boldsymbol{z}))) \longrightarrow \min$$

GAN

Objectives

$\log(D(G(\boldsymbol{z}))) \rightarrow \max$ or $\log(1 - D(G(\boldsymbol{z}))) \rightarrow \min$, that is the question.

In publications:

$$\min_G \max_D J(D, G) = \min_G \max_D \mathbb{E}_{\boldsymbol{x} \sim p_X(\boldsymbol{x})} [\log(D(\boldsymbol{x}))] + \mathbb{E}_{\boldsymbol{z} \sim p_Z(\boldsymbol{z})} [\log(1 - D(G(\boldsymbol{z})))]$$

- Pro: Adversarial objectives in one function as min-max game.
- Pro: Sometimes simultaneous minimization and maximization.
- Con: Numerically suboptimal, i.e. saturation with weak gradients early in training (NS-GAN Fedus et al. 2017):

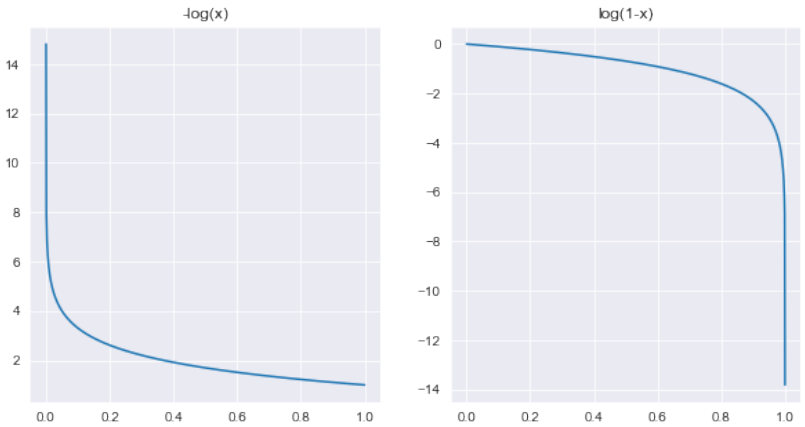


Figure: Comparison of two variants of J_G

GAN

Training

The training algorithm: For each training epoch do:

1. sample seeds $\{z_i\}_{i=1}^m$ from known $p_Z(z)$.
2. sample data $\{x_i\}_{i=1}^m$ from empirical distribution $p_X(x)$, i.e. the data set.
3. update D by ascending the gradient of J w.r.t. θ_D:

$$\theta_D = \theta_D + \frac{\partial}{\partial \theta_D} \frac{1}{m} \sum_{i=1}^{m} \log(D(x_i)) + \log(1 - D(G(z_i)))$$

4. update G by descending the gradient of J w.r.t. θ_G:

$$\theta_G = \theta_G - \frac{\partial}{\partial \theta_G} \frac{1}{m} \sum_{i=1}^{m} \log(1 - D(G(z_i)))$$

In practice though, the update of G is often implemented as

$$\theta_G = \theta_G + \frac{\partial}{\partial \theta_G} \frac{1}{m} \sum_{i=1}^{m} \log(D(G(z_i)))$$

GAN

Implementation in TF

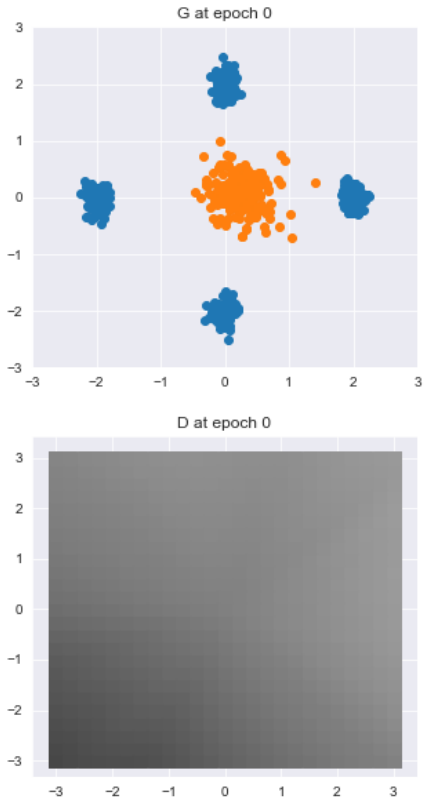
```
# Objective functions: ##########################
D_r = discriminator(X)
D_g = discriminator(generator(Z))

J_D_r = tf.reduce_mean(
    tf.nn.sigmoid_cross_entropy_with_logits(
        logits=D_r,
        labels=tf.ones_like(D_r),
    )
)
J_D_g = tf.reduce_mean(
    tf.nn.sigmoid_cross_entropy_with_logits(
        logits=D_g,
        labels=tf.zeros_like(D_g),
    )
)
J_D = J_D_r + J_D_g
J_G = tf.reduce_mean(
    tf.nn.sigmoid_cross_entropy_with_logits(
        logits=D_g,
        labels=tf.ones_like(D_g),
    )
)
```

```
# Optimization: ################################
D_solver = tf.train.AdamOptimizer(lr).minimize(
    J_D,
    var_list=theta_D
)
G_solver = tf.train.AdamOptimizer(lr).minimize(
    J_G,
    var_list=theta_G
)
for l in range(n_epochs):
    _, ep_D_loss = sess.run([D_solver, J_D],
        feed_dict={X: X_mb, Z:Z_mb}
    )
    _, ep_G_loss = sess.run([G_solver, J_G],
        feed_dict={Z: Z_mb}
    )
```

GAN

Demo



GAN

Theory

$$J(G, D) = \mathbb{E}_{\boldsymbol{x} \sim p_X(\boldsymbol{x})} [\log(D(\boldsymbol{x}))] + \mathbb{E}_{\boldsymbol{z} \sim p_Z(\boldsymbol{z})} [\log(1 - D(G(\boldsymbol{z})))]$$
$$= \mathbb{E}_{\boldsymbol{x} \sim p_X(\boldsymbol{x})} [\log(D(\boldsymbol{x}))] + \mathbb{E}_{\boldsymbol{x} \sim p_g(\boldsymbol{x})} [\log(1 - D(\boldsymbol{x}))]$$
$$= \int_x p_X(\boldsymbol{x}) \log(D(\boldsymbol{x})) d\boldsymbol{x} + \int_x p_g(\boldsymbol{x}) \log(1 - D(\boldsymbol{x})) d\boldsymbol{x}$$
$$= \int_x p_X(\boldsymbol{x}) \log(D(\boldsymbol{x})) + p_g(\boldsymbol{x}) \log(1 - D(\boldsymbol{x})) d\boldsymbol{x}$$

$$\Rightarrow \arg\max_D J(G, D) = \frac{p_X(\boldsymbol{x})}{p_X(\boldsymbol{x}) + p_g(\boldsymbol{x})} \tag{2}$$

for $\arg\max_x f(x) = \frac{a}{a+b}$ for $f(x) = a \cdot \log(x) + b \cdot \log(1 - x)$

- For any generator, the optimal generator $D^*(\boldsymbol{x})$ takes the value of Eq. (2).
- For an optimal generator, $D^*(\boldsymbol{x}) = \frac{1}{2}$

GAN

Theory

$$
\begin{aligned}
J(G, D) &= \mathbb{E}_{\boldsymbol{x} \sim p_X(\boldsymbol{x})} [\log(D(\boldsymbol{x}))] + \mathbb{E}_{\boldsymbol{z} \sim p_Z(\boldsymbol{z})} [\log(1 - D(G(\boldsymbol{z})))] \\
&= \mathbb{E}_{\boldsymbol{x} \sim p_X(\boldsymbol{x})} [\log(D(\boldsymbol{x}))] + \mathbb{E}_{\boldsymbol{x} \sim p_g(\boldsymbol{x})} [\log(1 - D(\boldsymbol{x}))] \\
&= \int_x p_X(\boldsymbol{x}) \log(D(\boldsymbol{x})) d\boldsymbol{x} + \int_x p_g(\boldsymbol{x}) \log(1 - D(\boldsymbol{x})) d\boldsymbol{x} \\
&= \int_x p_X(\boldsymbol{x}) \log(D(\boldsymbol{x})) + p_g(\boldsymbol{x}) \log(1 - D(\boldsymbol{x})) d\boldsymbol{x}
\end{aligned}
$$

$$
\Rightarrow \arg\max_D J(G, D) = \frac{p_X(\boldsymbol{x})}{p_X(\boldsymbol{x}) + p_g(\boldsymbol{x})} \tag{2}
$$

for $\arg\max_x f(x) = \frac{a}{a+b}$ for $f(x) = a \cdot \log(x) + b \cdot \log(1 - x)$

- For any generator, the optimal generator $D^*(\boldsymbol{x})$ takes the value of Eq. (2).
- For an optimal generator, $D^*(\boldsymbol{x}) = \frac{1}{2}$

GAN

Theory

$$
\begin{aligned}
J(G, D) &= \mathbb{E}_{\boldsymbol{x} \sim p_X(\boldsymbol{x})} [\log(D(\boldsymbol{x}))] + \mathbb{E}_{\boldsymbol{z} \sim p_Z(\boldsymbol{z})} [\log(1 - D(G(\boldsymbol{z})))] \\
&= \mathbb{E}_{\boldsymbol{x} \sim p_X(\boldsymbol{x})} [\log(D(\boldsymbol{x}))] + \mathbb{E}_{\boldsymbol{x} \sim p_g(\boldsymbol{x})} [\log(1 - D(\boldsymbol{x}))] \\
&= \int_x p_X(\boldsymbol{x}) \log(D(\boldsymbol{x})) d\boldsymbol{x} + \int_x p_g(\boldsymbol{x}) \log(1 - D(\boldsymbol{x})) d\boldsymbol{x} \\
&= \int_x p_X(\boldsymbol{x}) \log(D(\boldsymbol{x})) + p_g(\boldsymbol{x}) \log(1 - D(\boldsymbol{x})) d\boldsymbol{x}
\end{aligned}
$$

$$
\Rightarrow \arg\max_D J(G, D) = \frac{p_X(\boldsymbol{x})}{p_X(\boldsymbol{x}) + p_g(\boldsymbol{x})} \tag{2}
$$

for $\arg\max_x f(x) = \frac{a}{a+b}$ for $f(x) = a \cdot \log(x) + b \cdot \log(1 - x)$

- For any generator, the optimal generator $D^*(\boldsymbol{x})$ takes the value of Eq. (2).
- For an optimal generator, $D^*(\boldsymbol{x}) = \frac{1}{2}$

GAN

Theory

$$
\begin{aligned}
J(G, D) &= \mathbb{E}_{\boldsymbol{x} \sim p_X(\boldsymbol{x})} [\log(D(\boldsymbol{x}))] + \mathbb{E}_{\boldsymbol{z} \sim p_Z(\boldsymbol{z})} [\log(1 - D(G(\boldsymbol{z})))] \\
&= \mathbb{E}_{\boldsymbol{x} \sim p_X(\boldsymbol{x})} [\log(D(\boldsymbol{x}))] + \mathbb{E}_{\boldsymbol{x} \sim p_g(\boldsymbol{x})} [\log(1 - D(\boldsymbol{x}))] \\
&= \int_x p_X(\boldsymbol{x}) \log(D(\boldsymbol{x})) d\boldsymbol{x} + \int_x p_g(\boldsymbol{x}) \log(1 - D(\boldsymbol{x})) d\boldsymbol{x} \\
&= \int_x p_X(\boldsymbol{x}) \log(D(\boldsymbol{x})) + p_g(\boldsymbol{x}) \log(1 - D(\boldsymbol{x})) d\boldsymbol{x}
\end{aligned}
$$

$$
\Rightarrow \arg\max_D J(G, D) = \frac{p_X(\boldsymbol{x})}{p_X(\boldsymbol{x}) + p_g(\boldsymbol{x})} \tag{2}
$$

for $\arg\max_x f(x) = \frac{a}{a+b}$ for $f(x) = a \cdot \log(x) + b \cdot \log(1 - x)$

- For any generator, the optimal generator $D^*(\boldsymbol{x})$ takes the value of Eq. (2).
- For an optimal generator, $D^*(\boldsymbol{x}) = \frac{1}{2}$

GAN

Theory

$$
\begin{aligned}
J(G, D) &= \mathbb{E}_{\boldsymbol{x} \sim p_X(\boldsymbol{x})} [\log(D(\boldsymbol{x}))] + \mathbb{E}_{\boldsymbol{z} \sim p_Z(\boldsymbol{z})} [\log(1 - D(G(\boldsymbol{z})))] \\
&= \mathbb{E}_{\boldsymbol{x} \sim p_X(\boldsymbol{x})} [\log(D(\boldsymbol{x}))] + \mathbb{E}_{\boldsymbol{x} \sim p_g(\boldsymbol{x})} [\log(1 - D(\boldsymbol{x}))] \\
&= \int_x p_X(\boldsymbol{x}) \log(D(\boldsymbol{x})) d\boldsymbol{x} + \int_x p_g(\boldsymbol{x}) \log(1 - D(\boldsymbol{x})) d\boldsymbol{x} \\
&= \int_x p_X(\boldsymbol{x}) \log(D(\boldsymbol{x})) + p_g(\boldsymbol{x}) \log(1 - D(\boldsymbol{x})) d\boldsymbol{x}
\end{aligned}
$$

$$
\Rightarrow \arg\max_D J(G, D) = \frac{p_X(\boldsymbol{x})}{p_X(\boldsymbol{x}) + p_g(\boldsymbol{x})} \tag{2}
$$

for $\arg\max_x f(x) = \frac{a}{a+b}$ for $f(x) = a \cdot \log(x) + b \cdot \log(1 - x)$

- For any generator, the optimal generator $D^*(\boldsymbol{x})$ takes the value of Eq. (2).
- For an optimal generator, $D^*(\boldsymbol{x}) = \frac{1}{2}$

GAN

Theory

$$
\begin{aligned}
& JSD(p_X, p_g) \\
=& \frac{1}{2} KL\left(p_X || \frac{p_X + p_g}{2}\right) + \frac{1}{2} KL\left(p_g || \frac{p_X + p_g}{2}\right) && | \text{ Def. } JSD \\
=& \frac{1}{2}\left(\int_x p_X(x) \log\left(\frac{2p_X(x)}{p_X(x) + p_g(x)}\right) + p_g(x) \log\left(\frac{2p_g(x)}{p_X(x) + p_g(x)}\right) dx\right) && | \text{ Def. } KL \\
=& \frac{1}{2}\left(\log(4) + \int_x p_X(x) \log\left(\frac{p_X(x)}{p_X(x) + p_g(x)}\right) + p_g(x) \log\left(1 - \frac{p_X(x)}{p_X(x) + p_g(x)}\right) dx\right) \\
=& \frac{1}{2}\left(\log(4) + \mathbb{E}_{x \sim p_X} \log(D^*(x)) + \mathbb{E}_{x \sim p_g} \log(1 - D^*(x))\right) && | \text{ Eq.(2)} \\
=& \frac{1}{2}\left(\log(4) + J(G, D^*)\right) && | \text{ Def. } J(G, D) \\
\Rightarrow & J(G, D^*) = -\log(4) + 2 \cdot JSD(p_X || p_g)
\end{aligned}
$$

The optimal generator minimizes the Jensen-Shannon Divergence between p_X and p_g.
In case of optimal G^* as well, $J(G^*, D^*) = -\log(4)$.

GAN

Theory

$$
\begin{aligned}
& JSD(p_X, p_g) \\
=& \frac{1}{2} KL\left(p_X || \frac{p_X + p_g}{2}\right) + \frac{1}{2} KL\left(p_g || \frac{p_X + p_g}{2}\right) && | \text{ Def. } JSD \\
=& \frac{1}{2}\left(\int_x p_X(\boldsymbol{x}) \log\left(\frac{2p_X(\boldsymbol{x})}{p_X(\boldsymbol{x}) + p_g(\boldsymbol{x})}\right) + p_g(\boldsymbol{x}) \log\left(\frac{2p_g(\boldsymbol{x})}{p_X(\boldsymbol{x}) + p_g(\boldsymbol{x})}\right) d\boldsymbol{x}\right) && | \text{ Def. } KL \\
=& \frac{1}{2}\left(\log(4) + \int_x p_X(\boldsymbol{x}) \log\left(\frac{p_X(\boldsymbol{x})}{p_X(\boldsymbol{x}) + p_g(\boldsymbol{x})}\right) + p_g(\boldsymbol{x}) \log\left(1 - \frac{p_X(\boldsymbol{x})}{p_X(\boldsymbol{x}) + p_g(\boldsymbol{x})}\right) d\boldsymbol{x}\right) \\
=& \frac{1}{2}\left(\log(4) + \mathbb{E}_{x \sim p_X} \log(D^*(\boldsymbol{x})) + \mathbb{E}_{x \sim p_g} \log(1 - D^*(\boldsymbol{x}))\right) && | \text{ Eq.(2)} \\
=& \frac{1}{2}\left(\log(4) + J(G, D^*)\right) && | \text{ Def. } J(G, D) \\
\Rightarrow & J(G, D^*) = -\log(4) + 2 \cdot JSD(p_X || p_g)
\end{aligned}
$$

The optimal generator minimizes the Jensen-Shannon Divergence between p_X and p_g.
In case of optimal G^* as well, $J(G^*, D^*) = -\log(4)$.

GAN

Theory

$$
\begin{aligned}
& JSD(p_X, p_g) \\
=& \frac{1}{2} KL\left(p_X || \frac{p_X + p_g}{2}\right) + \frac{1}{2} KL\left(p_g || \frac{p_X + p_g}{2}\right) && | \text{ Def. } JSD \\
=& \frac{1}{2}\left(\int_x p_X(\boldsymbol{x}) \log\left(\frac{2p_X(\boldsymbol{x})}{p_X(\boldsymbol{x}) + p_g(\boldsymbol{x})}\right) + p_g(\boldsymbol{x}) \log\left(\frac{2p_g(\boldsymbol{x})}{p_X(\boldsymbol{x}) + p_g(\boldsymbol{x})}\right) d\boldsymbol{x}\right) && | \text{ Def. } KL \\
=& \frac{1}{2}\left(\log(4) + \int_x p_X(\boldsymbol{x}) \log\left(\frac{p_X(\boldsymbol{x})}{p_X(\boldsymbol{x}) + p_g(\boldsymbol{x})}\right) + p_g(\boldsymbol{x}) \log\left(1 - \frac{p_X(\boldsymbol{x})}{p_X(\boldsymbol{x}) + p_g(\boldsymbol{x})}\right) d\boldsymbol{x}\right) \\
=& \frac{1}{2}\left(\log(4) + \mathbb{E}_{x \sim p_X} \log(D^*(\boldsymbol{x})) + \mathbb{E}_{x \sim p_g} \log(1 - D^*(\boldsymbol{x}))\right) && | \text{ Eq.(2)} \\
=& \frac{1}{2}\left(\log(4) + J(G, D^*)\right) && | \text{ Def. } J(G, D) \\
\Rightarrow & J(G, D^*) = -\log(4) + 2 \cdot JSD(p_X || p_g)
\end{aligned}
$$

The optimal generator minimizes the Jensen-Shannon Divergence between p_X and p_g.
In case of optimal G^* as well, $J(G^*, D^*) = -\log(4)$.

GAN

Theory

$$
\begin{aligned}
& JSD(p_X, p_g) \\
=& \frac{1}{2} KL\left(p_X || \frac{p_X + p_g}{2}\right) + \frac{1}{2} KL\left(p_g || \frac{p_X + p_g}{2}\right) && | \text{ Def. } JSD \\
=& \frac{1}{2}\left(\int_x p_X(\boldsymbol{x}) \log\left(\frac{2p_X(\boldsymbol{x})}{p_X(\boldsymbol{x}) + p_g(\boldsymbol{x})}\right) + p_g(\boldsymbol{x}) \log\left(\frac{2p_g(\boldsymbol{x})}{p_X(\boldsymbol{x}) + p_g(\boldsymbol{x})}\right) d\boldsymbol{x}\right) && | \text{ Def. } KL \\
=& \frac{1}{2}\left(\log(4) + \int_x p_X(\boldsymbol{x}) \log\left(\frac{p_X(\boldsymbol{x})}{p_X(\boldsymbol{x}) + p_g(\boldsymbol{x})}\right) + p_g(\boldsymbol{x}) \log\left(1 - \frac{p_X(\boldsymbol{x})}{p_X(\boldsymbol{x}) + p_g(\boldsymbol{x})}\right) d\boldsymbol{x}\right) \\
=& \frac{1}{2}\left(\log(4) + \mathbb{E}_{x \sim p_X} \log(D^*(\boldsymbol{x})) + \mathbb{E}_{x \sim p_g} \log(1 - D^*(\boldsymbol{x}))\right) && | \text{ Eq.(2)} \\
=& \frac{1}{2}\left(\log(4) + J(G, D^*)\right) && | \text{ Def. } J(G, D) \\
\Rightarrow & J(G, D^*) = -\log(4) + 2 \cdot JSD(p_X || p_g)
\end{aligned}
$$

The optimal generator minimizes the Jensen-Shannon Divergence between p_X and p_g.
In case of optimal G^* as well, $J(G^*, D^*) = -\log(4)$.

GAN

Theory

$$
\begin{aligned}
& JSD(p_X, p_g) \\
=& \frac{1}{2} KL\left(p_X || \frac{p_X + p_g}{2}\right) + \frac{1}{2} KL\left(p_g || \frac{p_X + p_g}{2}\right) && | \text{ Def. } JSD \\
=& \frac{1}{2}\left(\int_x p_X(\boldsymbol{x}) \log\left(\frac{2p_X(\boldsymbol{x})}{p_X(\boldsymbol{x}) + p_g(\boldsymbol{x})}\right) + p_g(\boldsymbol{x}) \log\left(\frac{2p_g(\boldsymbol{x})}{p_X(\boldsymbol{x}) + p_g(\boldsymbol{x})}\right) d\boldsymbol{x}\right) && | \text{ Def. } KL \\
=& \frac{1}{2}\left(\log(4) + \int_x p_X(\boldsymbol{x}) \log\left(\frac{p_X(\boldsymbol{x})}{p_X(\boldsymbol{x}) + p_g(\boldsymbol{x})}\right) + p_g(\boldsymbol{x}) \log\left(1 - \frac{p_X(\boldsymbol{x})}{p_X(\boldsymbol{x}) + p_g(\boldsymbol{x})}\right) d\boldsymbol{x}\right) \\
=& \frac{1}{2}\left(\log(4) + \mathbb{E}_{x \sim p_X} \log(D^*(\boldsymbol{x})) + \mathbb{E}_{x \sim p_g} \log(1 - D^*(\boldsymbol{x}))\right) && | \text{ Eq.(2)} \\
=& \frac{1}{2}\left(\log(4) + J(G, D^*)\right) && | \text{ Def. } J(G, D) \\
\Rightarrow & J(G, D^*) = -\log(4) + 2 \cdot JSD(p_X || p_g)
\end{aligned}
$$

The optimal generator minimizes the Jensen-Shannon Divergence between p_X and p_g.
In case of optimal G^* as well, $J(G^*, D^*) = -\log(4)$.

GAN

Theory

$$
\begin{aligned}
& JSD(p_X, p_g) \\
=& \frac{1}{2} KL\left(p_X || \frac{p_X + p_g}{2}\right) + \frac{1}{2} KL\left(p_g || \frac{p_X + p_g}{2}\right) && | \text{ Def. } JSD \\
=& \frac{1}{2}\left(\int_{\boldsymbol{x}} p_X(\boldsymbol{x}) \log\left(\frac{2p_X(\boldsymbol{x})}{p_X(\boldsymbol{x}) + p_g(\boldsymbol{x})}\right) + p_g(\boldsymbol{x}) \log\left(\frac{2p_g(\boldsymbol{x})}{p_X(\boldsymbol{x}) + p_g(\boldsymbol{x})}\right) d\boldsymbol{x}\right) && | \text{ Def. } KL \\
=& \frac{1}{2}\left(\log(4) + \int_{\boldsymbol{x}} p_X(\boldsymbol{x}) \log\left(\frac{p_X(\boldsymbol{x})}{p_X(\boldsymbol{x}) + p_g(\boldsymbol{x})}\right) + p_g(\boldsymbol{x}) \log\left(1 - \frac{p_X(\boldsymbol{x})}{p_X(\boldsymbol{x}) + p_g(\boldsymbol{x})}\right) d\boldsymbol{x}\right) \\
=& \frac{1}{2}\left(\log(4) + \mathbb{E}_{x \sim p_X} \log(D^*(\boldsymbol{x})) + \mathbb{E}_{x \sim p_g} \log(1 - D^*(\boldsymbol{x}))\right) && | \text{ Eq.(2)} \\
=& \frac{1}{2}\left(\log(4) + J(G, D^*)\right) && | \text{ Def. } J(G, D) \\
\Rightarrow & J(G, D^*) = -\log(4) + 2 \cdot JSD(p_X || p_g)
\end{aligned}
$$

The optimal generator minimizes the Jensen-Shannon Divergence between p_X and p_g.
In case of optimal G^* as well, $J(G^*, D^*) = -\log(4)$.

CGAN

Definition

[Mirza and Osindero 2014] Conditional GAN (CGAN) generates data sample based on labels.

- Labeled training samples: $(\boldsymbol{x}, \boldsymbol{y})$
- Generator: $G(\boldsymbol{x}, \boldsymbol{y})$
- Discriminator: $D(\boldsymbol{x}, \boldsymbol{y}) = \mathbb{P}(\boldsymbol{x} \sim p_X(\boldsymbol{x})|\boldsymbol{y})$
- Objective:

$$\min_G \max_D J(D, G) = \mathbb{E}_{\boldsymbol{x} \sim p_X(\boldsymbol{x})} [\log(D(\boldsymbol{x}, y))] + \mathbb{E}_{\boldsymbol{z} \sim p_Z(\boldsymbol{z})} [\log(1 - D(G(\boldsymbol{z}, \boldsymbol{y}), \boldsymbol{y}))]$$

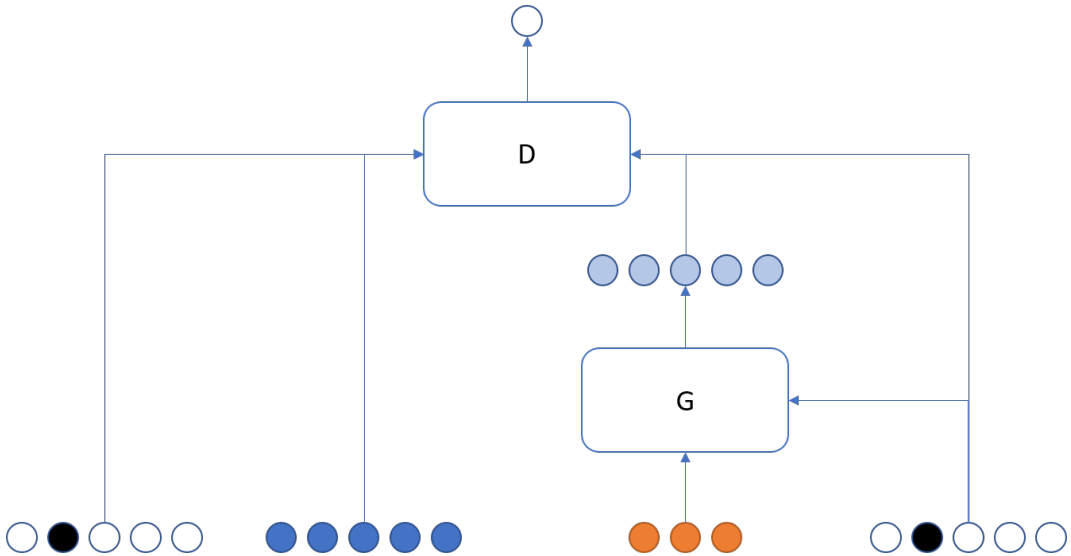


Figure: Architecture of CGAN

CGAN

Implementation in TF

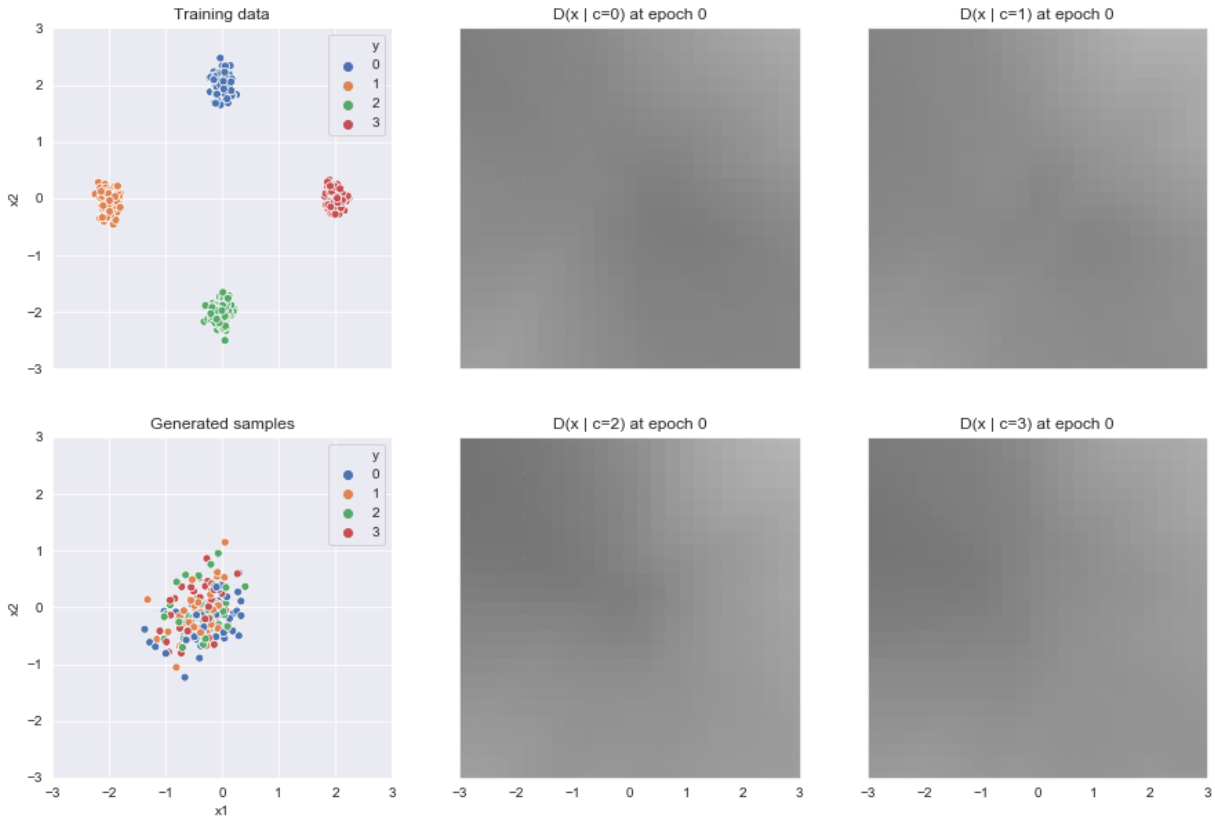
```
# Define the D: ###################################
def discriminator(x, c):
    D_h1 = tf.nn.relu(tf.matmul(tf.concat([x, c], axis=1), D_W1) + D_b1)
    D_h2 = tf.nn.relu(tf.matmul(D_h1, D_W2) + D_b2)
    D_out = tf.matmul(D_h2, D_W3) + D_b3
    return D_out

# Define the G: ###################################
def generator(z, c):
    G_h1 = tf.nn.relu(tf.matmul(tf.concat([z, c], axis=1), G_W1) + G_b1)
    G_h2 = tf.nn.relu(tf.matmul(G_h1, G_W2) + G_b2)
    G_out = tf.matmul(G_h2, G_W3) + G_b3
    return G_out

# Objectives: #####################################
D_r = discriminator(X, C)
D_f = discriminator(generator(Z, C), C)
```

CGAN

Demo


Training data
D(x | c=0) at epoch 0
D(x | c=1) at epoch 0
Generated samples
D(x | c=2) at epoch 0
D(x | c=3) at epoch 0
y
0
1
2
3
x1
x2

InfoGAN

Definition

[Chen et al. 2016] No labels available. But one assumes k clusters that can be described with a multinomial (categorical) distribution with $M(1, (p_1, p_2, ..., p_k))$

- Unlabeld training samples: $(\boldsymbol{x})$.
- Generator: $G(\boldsymbol{z}, c), c \sim M(1, (p_1, p_2, ..., p_k))$, where c is known as the latent code.
- Discriminator: $D(\boldsymbol{x}) = \mathbb{P}(\boldsymbol{x} \sim p_X(\boldsymbol{x}))$
- Info-function, or a second discriminator:

$$Q(\boldsymbol{x}) = \mathbb{P}(\boldsymbol{x} \text{ of cluster } k | \boldsymbol{x} \sim p_{model}(\boldsymbol{x}))$$

$\Rightarrow$ An additional objective: $h(c, Q(G(\boldsymbol{z}, c)))$ w.r.t. G and Q.

InfoGAN

Definition

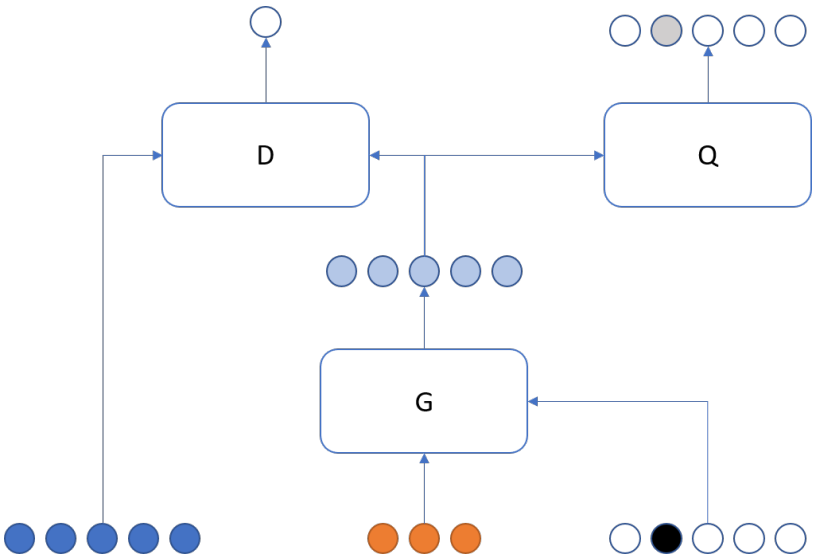


Figure: Architecure of an InfoGAN

- The path $c \to G \to Q \to \hat{c}$ forms an AE for c.
- $G(z, c)$ is thus guaranteed to contain information of c.
- The code c could be continuous as well as discrete.

InfoGAN

Definition

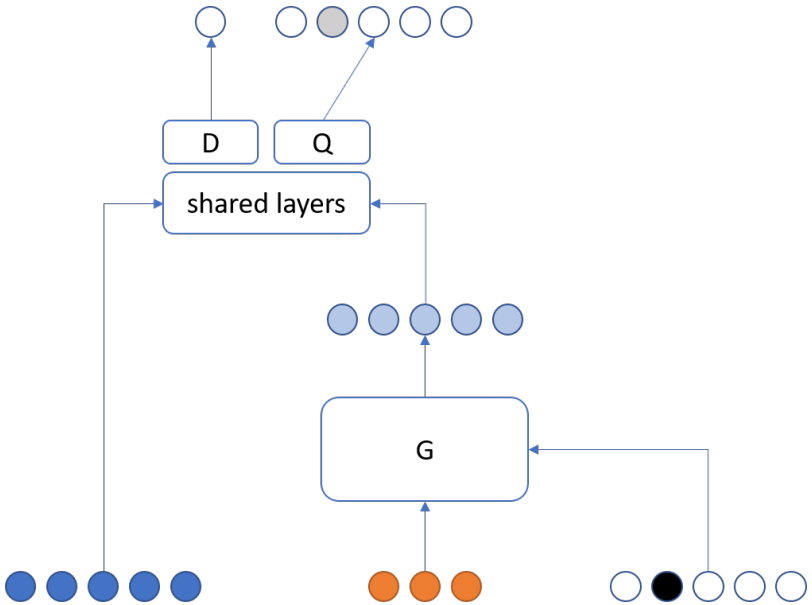


Figure: Architecure of an InfoGAN

- The path $c \to G \to Q \to \hat{c}$ forms an AE for c.
- $G(z, c)$ is thus guaranteed to contain information of c.
- The code c could be continuous as well as discrete.

InfoGAN

Implementation in TF

```
# Define the D: ######################################
def discriminator(x):
    ...  # same as GAN

# Define the G: ######################################
def generator(z, c):
    ...  # same as GAN

# Define the Q: ######################################
def q_info(x):
    Q_h1 = tf.nn.relu(tf.matmul(x, Q_W1) + Q_b1)
    Q_logits = tf.matmul(Q_h1, Q_W2) + Q_b2
    return Q_logits

# Sampling functions: ################################
def sample_c(m):
    return np.random.multinomial(1, 4*[0.25], size=m)

def sample_Z(m, n):
    return np.random.randn(m, n)
```

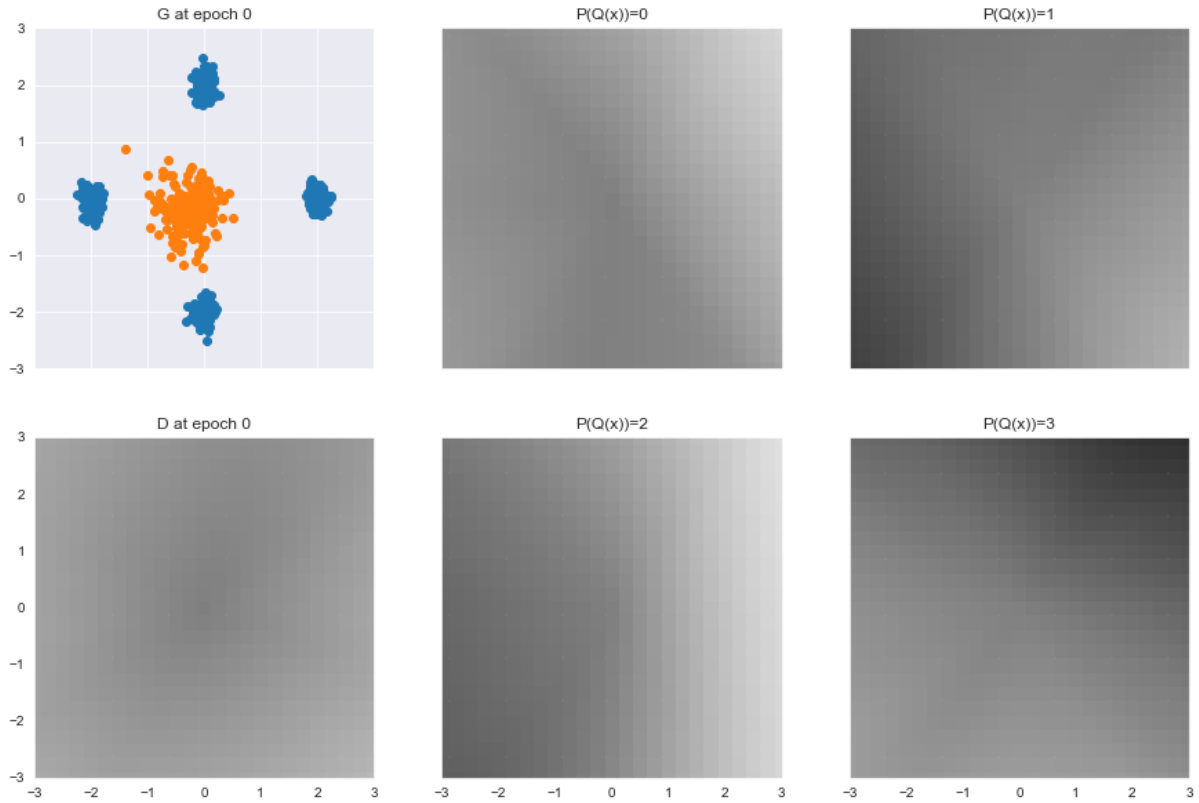
```
# Objectives: ########################################
D_r = discriminator(X)
D_g = discriminator(generator(Z, C))
Q_g = q_info(generator(Z, C))

D_loss = ...  # same as GAN
G_loss = ...  # same as CGAN
Q_loss = tf.reduce_mean(
    tf.nn.softmax_cross_entropy_with_logits_v2(
        labels=C,
        logits=Q_g,
    )
)

D_solver = ...  # same as GAN
G_solver = ...  # same as GAN
Q_solver = tf.train.AdamOptimizer(1e-4).minimize(
    Q_loss,
    var_list=theta_G + theta_Q,
)
```

InfoGAN

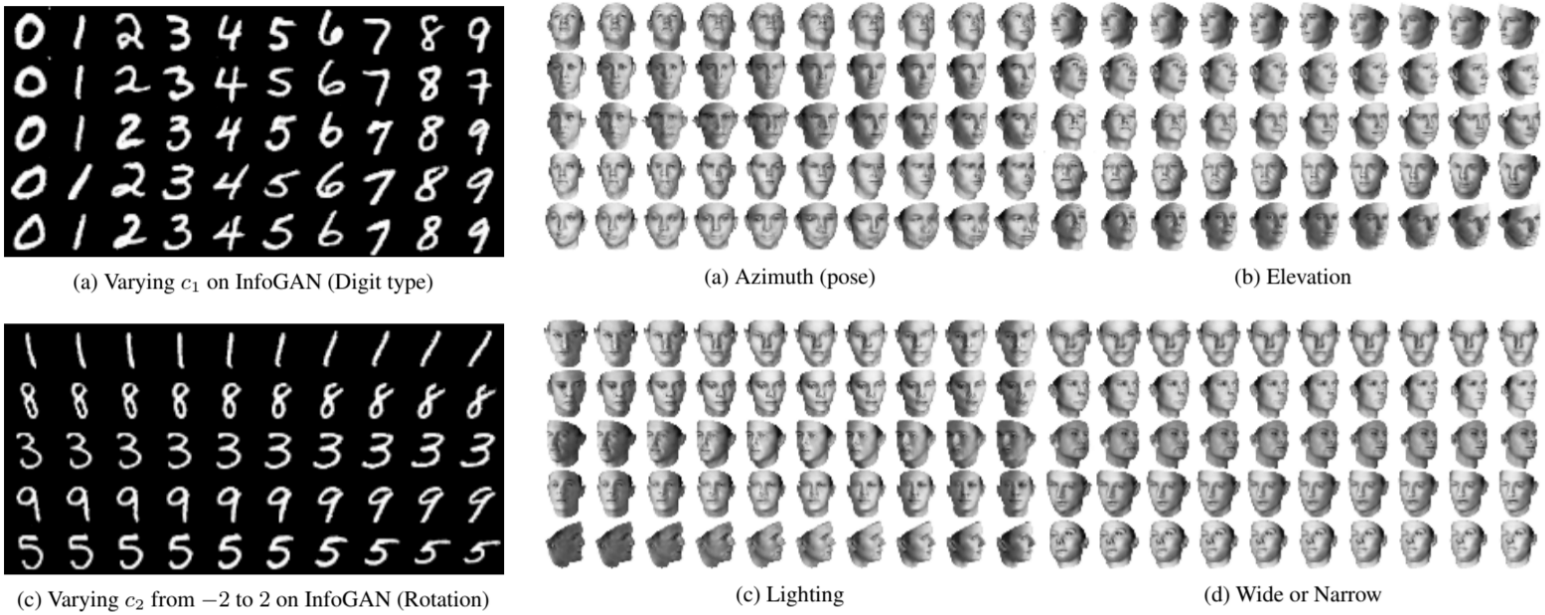
Demo



InfoGAN

Examples

The code c could also be numerical.



(a) Varying c_1 on InfoGAN (Digit type)

(a) Azimuth (pose)

(b) Elevation

(c) Varying c_2 from -2 to 2 on InfoGAN (Rotation)

(c) Lighting

(d) Wide or Narrow

Figure: Examples from Chen et al. 2016

CycleGAN

Definition

[Zhu et al. 2017]: A more general case: mapping from an unknown distribution $p_Z(\boldsymbol{z})$ which is only represented by a dataset.

- *Unpaired* training samples from two domains:

$$(\boldsymbol{x}_i)_{i=1}^N, \boldsymbol{x}_i \in \mathcal{A}$$
$$(\boldsymbol{z}_i)_{i=1}^M, \boldsymbol{z}_j \in \mathcal{B}.$$

- Discriminators:

$$D^{\mathcal{A}}(\boldsymbol{x}) = \mathbb{P}(\boldsymbol{x} \sim p_X(\boldsymbol{x}))$$
$$D^{\mathcal{B}}(\boldsymbol{z}) = \mathbb{P}(\boldsymbol{z} \sim p_Z(\boldsymbol{z}))$$

- Generators:

$$G^{\mathcal{A}}(\boldsymbol{z}) \in \mathcal{A}$$
$$G^{\mathcal{B}}(\boldsymbol{x}) \in \mathcal{B}$$

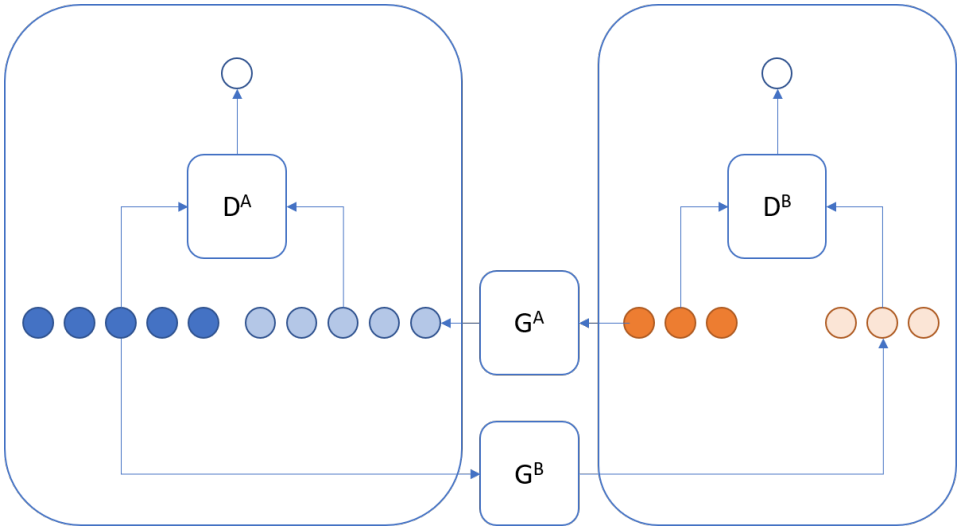


Figure: Cycle GAN architecture

- $\boldsymbol{z}\widehat{=}$ seeds $\Rightarrow$ learning representations.
- $\boldsymbol{z}\widehat{=}$ data from another domain $\Rightarrow$ learning a transformation.

CycleGAN

Objectives

$$J_D^{\mathcal{A}} = \log(D^{\mathcal{A}}(\boldsymbol{x})) + \log(1 - D^{\mathcal{A}}(G^{\mathcal{A}}(\boldsymbol{z}))) \quad \rightarrow \max$$
$$J_G^{\mathcal{A}} = \log(D^{\mathcal{A}}(G^{\mathcal{A}}(\boldsymbol{z}))) \quad \rightarrow \max$$
$$J_{cyc}^{\mathcal{A}} = ||\boldsymbol{x} - G^{\mathcal{A}}(G^{\mathcal{B}}(\boldsymbol{x}))||_1 \quad \rightarrow \min$$

$$J_D^{\mathcal{B}} = \log(D^{\mathcal{B}}(\boldsymbol{z})) + \log(1 - D^{\mathcal{B}}(G^{\mathcal{B}}(\boldsymbol{x}))) \quad \rightarrow \max$$
$$J_G^{\mathcal{B}} = \log(D^{\mathcal{B}}(G^{\mathcal{B}}(\boldsymbol{x}))) \quad \rightarrow \max$$
$$J_{cyc}^{\mathcal{B}} = ||\boldsymbol{z} - G^{\mathcal{B}}(G^{\mathcal{A}}(\boldsymbol{z}))||_1 \quad \rightarrow \min$$

CycleGAN

Implementation in TF

```
# In domain A: ###############################
def discriminatorA(x):
    ...

def generatorB2A(x):
    ...

# In domain B: ###############################
def discriminatorB(x):
    ...

def generatorA2B(x):
    ...

# Objectives of D's: ########################
DA_r = discriminatorA(X_A)
DA_g = discriminatorA(generatorB2A(X_B))

DB_r = discriminatorB(X_B)
DB_g = discriminatorB(generatorA2B(X_A))

J_D_A = ...(DA_r, DA_g)
J_D_B = ...(DB_r, DB_g)
```

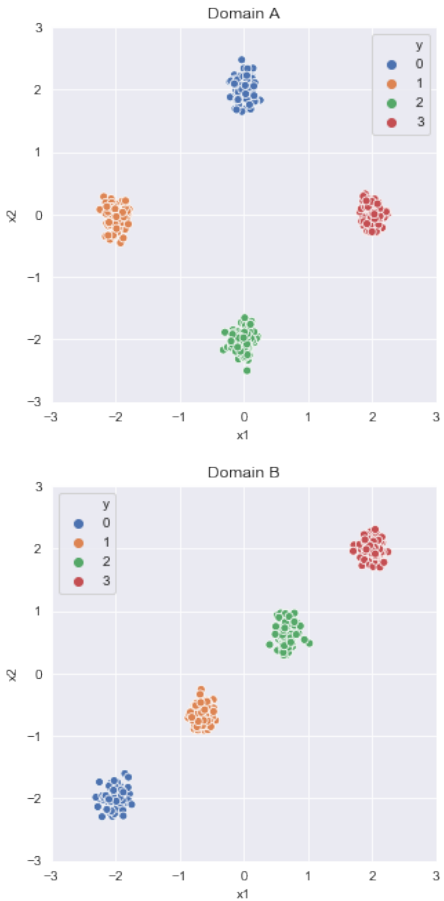
```
# Objectives of G's: ########################
J_A_cycle = tf.reduce_mean(
    tf.losses.absolute_difference(
        X_A,
        generatorB2A(generatorA2B(X_A))
    )
)
J_B_cycle = tf.reduce_mean(
    tf.losses.absolute_difference(
        X_B,
        generatorA2B(generatorB2A(X_B))
    )
)

J_G_A = tf.reduce_mean(
    tf.nn.sigmoid_cross_entropy_with_logits(
        logits=DA_g,
        labels=tf.ones_like(DA_g)
    )
)
J_G_A = J_G_A + J_A_cycle*lambd

J_G_B = tf.reduce_mean(
    tf.nn.sigmoid_cross_entropy_with_logits(
        logits=DB_g,
        labels=tf.ones_like(DB_g)
    )
)
J_G_B = J_G_B + J_B_cycle*lambd
```

CycleGAN

Demo



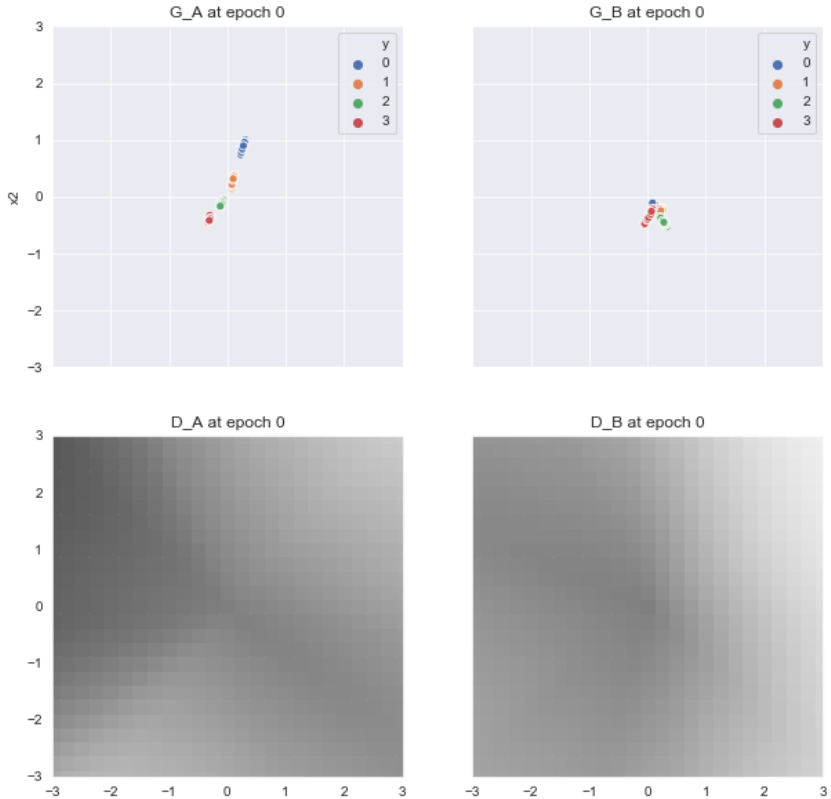


Figure: Training samples

CycleGAN

Examples

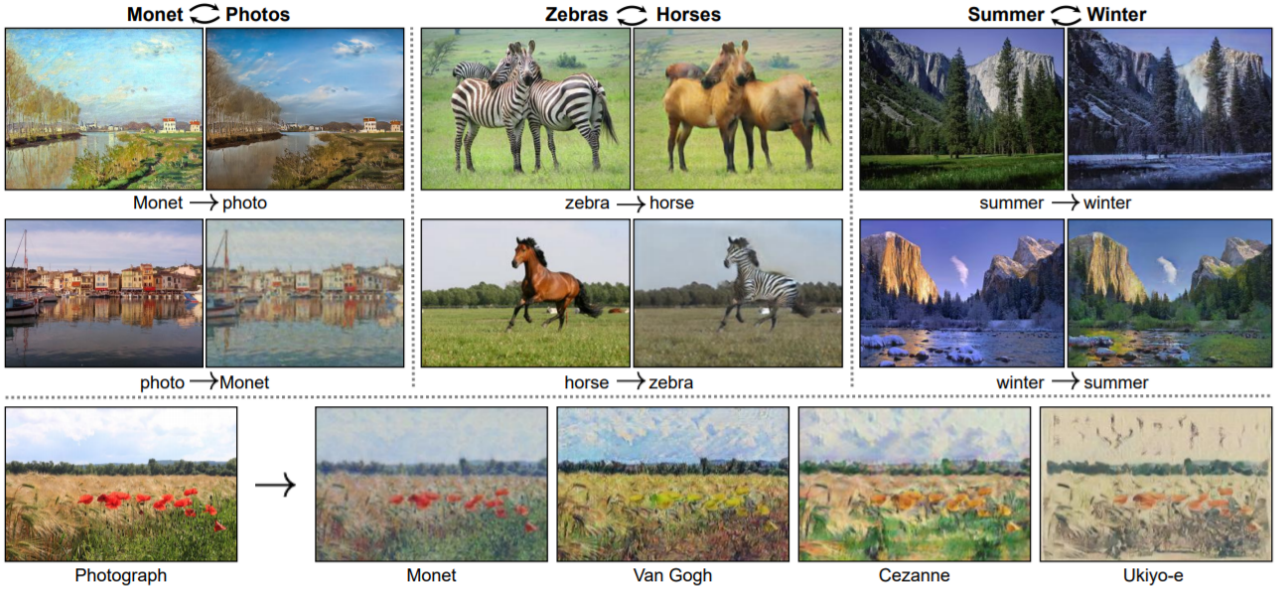


Figure 1: Given any two unordered image collections X and Y, our algorithm learns to automatically "translate" an image from one into the other and vice versa: *(left)* Monet paintings and landscape photos from Flickr; *(center)* zebras and horses from ImageNet; *(right)* summer and winter Yosemite photos from Flickr. Example application *(bottom)*: using a collection of paintings of famous artists, our method learns to render natural photographs into the respective styles.

Figure: Exmaples from Zhu et al. 2017

CycleGAN

Examples

Creative Applications of CycleGAN

Researchers, developers and artists have tried our code on various image manipulation and artistic creatiion tasks. Here we highlight a few of the many compelling examples. Search **CycleGAN** in Twitter for more applications.

Converting Monet into Thomas Kinkade

What if **Claude Monet** had lived to see the rise of Americana pastoral kitsch in the style of **Thomas Kinkade**? And what if he resorted to it to support himself in his old age? Using CycleGAN, our great **David Fouhey** finally realized the dream of Claude Monet revisiting his cherished work in light of Thomas Kinkade, the self-stylized painter of light.

Resurrecting Ancient Cities

Jack Clark used our code to convert ancient maps of **Babylon**, **Jerusalem** and **London** into modern Google Maps and satellite views.

Animal Transfiguration

Tatsuya Hatanaka trained our method to translate black bears to pandas. See more examples and download the models at the **website**. **Matt Powell** performed transfiguration between different species of birds

Portrait to Dollface

Mario Klingemann used our code to translate portraits into dollface. See how the characters in Game of Thrones look like in the doll world.

Face ↔ Ramen

Takuya Kato performed a magical and hilarious Face ↔ Ramen translation with CycleGAN. Check out more results **here**

Colorizing legacy photographs

Mario Klingemann trained our method to turn legacy black and white photos into color versions.

Figure: Exmaples from `https://junyanz.github.io/CycleGAN/`

Table of Contents

Fréchet Inception Distance Score

Given two samples $\boldsymbol{X}_r \in \mathbb{R}^{n_r \times p}$, $\boldsymbol{X}_g \in \mathbb{R}^{n_g \times p}$;
Assuming both samples follow Gaussian distributions;

$$FID(\boldsymbol{X}_r, \boldsymbol{X}_g) = ||\boldsymbol{\mu}_r - \boldsymbol{\mu}_g||_2^2 + tr\left(\boldsymbol{\Sigma}_r + \boldsymbol{\Sigma}_g - 2 \cdot (\boldsymbol{\Sigma}_r \boldsymbol{\Sigma}_g)^{\frac{1}{2}}\right)$$

where $\boldsymbol{\mu}_r, \boldsymbol{\mu}_g \in \mathbb{R}^p$ are the corresponding column-wise mean; $\boldsymbol{\Sigma}_r, \boldsymbol{\Sigma}_g \in \mathbb{R}^{p \times p}$ are the corresponding variance matrices.

- Left part: measuring the systematic shift / bias on average;
- Right part: measuring the difference in scales;
- Assume samples $\boldsymbol{A} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I}), \boldsymbol{B} \sim \mathcal{N}(\alpha + \boldsymbol{0}, \boldsymbol{I}), \boldsymbol{C} \sim \mathcal{N}(\boldsymbol{0}, \beta^2 \cdot \boldsymbol{I})$:

	Left part	Right part
$FID(\boldsymbol{A}, \boldsymbol{B})$	$\to p \cdot \alpha^2$	$\to 0$
$FID(\boldsymbol{A}, \boldsymbol{C})$	$\to 0$	$\to p \cdot (\beta - 1)^2$

KNN Score

Given two samples $\boldsymbol{X}_r \in \mathbb{R}^{n_r \times p}$, $\boldsymbol{X}_g \in \mathbb{R}^{n_g \times p}$;
Let $\boldsymbol{y}_r = \mathbf{1}$ and $\boldsymbol{y}_g = \mathbf{0}$ be labels.
Calculate the average Leave-One-Out accuracy of a 1-NN model on samples

$$\boldsymbol{X} = \begin{bmatrix} \boldsymbol{X}_r \\ \boldsymbol{X}_g \end{bmatrix}, \boldsymbol{y} = \begin{bmatrix} \boldsymbol{y}_r \\ \boldsymbol{y}_g \end{bmatrix},$$

I.e., for each $(\boldsymbol{x}_i, y_i)$, train a 1-NN model on $\boldsymbol{X}_{-i}, \boldsymbol{y}_{-i}$ and evaluate it on $(\boldsymbol{x}_i, y_i)$ in term of accuracy.

- KNN score $\rightarrow$ 0.5 if two sets of samples are similar and yet different;
- KNN score $\rightarrow$ 0 overfitting: one set is the copy of the other.
- KNN score $\rightarrow$ 1 different distributions.

Table of Contents

Evolution of generative models

2009

2015

2018

Figure: https://www.cs.toronto.edu/~rgrosse/courses/csc321_2018/slides/lec19.pdf

VAE v.s. GAN

	VAE	GANs
Training	More stable	Challenging in term of divergence
Multimodal data	Average generation	Sharper generation but mode collapse

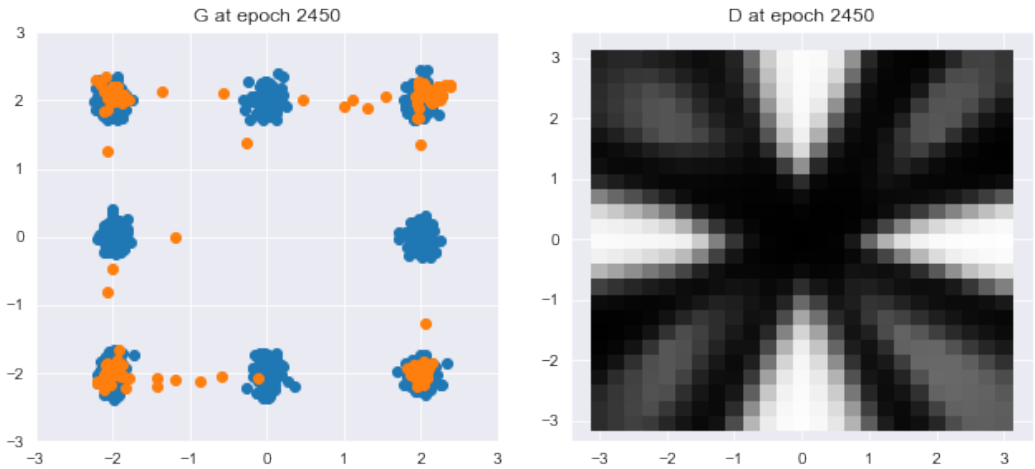


Figure: Illustration of mode collapse by GAN.

Discriminator getting too smart too fast (saturation) $\rightarrow$ divergence and mode collapse.

VAE v.s. GAN

Figure: Comparison of VAE (left and blurry) and GANs (right and sharp) generations, from `https://medium.com/@wuga/generate-anime-character-with-variational-auto-encoder-81e3134d1439`

VAE v.s. GAN

What does "blurry" actually mean?

$$J^{rec} = \sum_{i=1}^{N} ||\boldsymbol{x}_i - \widehat{\boldsymbol{x}}_i||_2^2$$

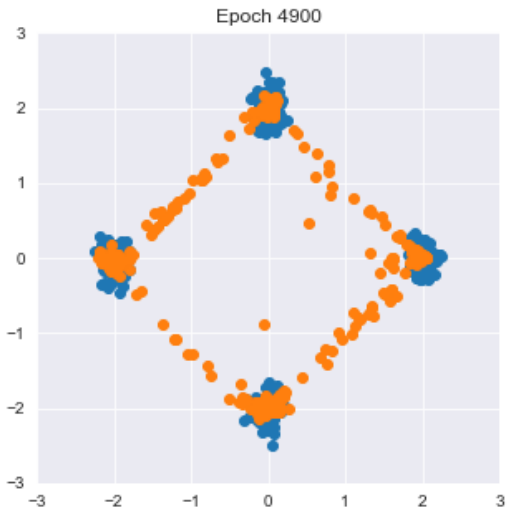


Figure: Illustration of blurry generation

A Zoo of GANs

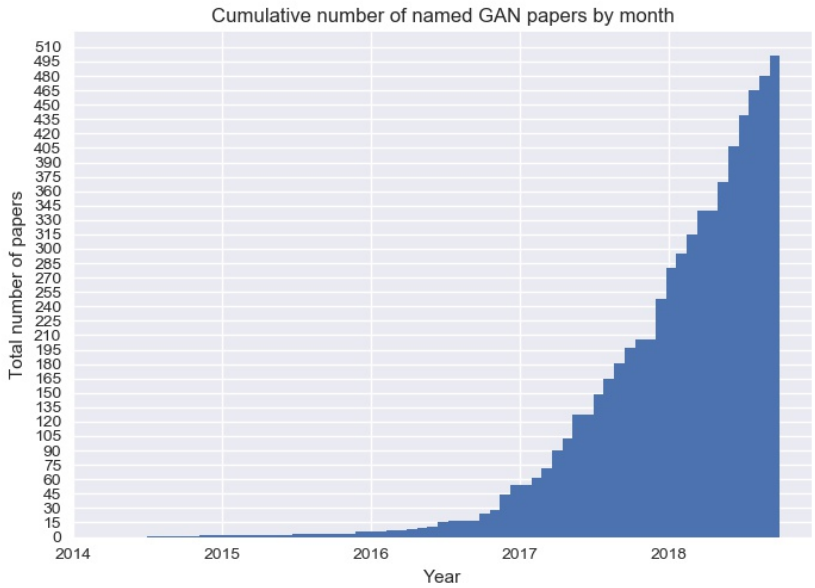


Figure: Statistics from `https://github.com/hindupuravinash/the-gan-zoo`

- New functionalities: CGAN, InfoGAN, CycleGAN, BiGAN, ...
- New use cases: DCGAN, SimGAN, SuperResolution, Progressive Growing, ...
- Better learning: WGAN, f-GAN, LSGAN, Fisher GAN, Unrolled GAN, ...
  Lucic et al. 2018: *Are GANs Created Equal? A Large-Scale Study*

A Zoo of GANs

We find that most models can reach similar scores with enough hyperparameter optimization and random restarts. This suggests that improvements can arise from a higher computational budget and tuning more than fundamental algorithmic changes. [...] Finally, we did not find evidence that any of the tested algorithms consistently outperforms the non-saturating GAN introduced in Goodfellow et al. 2014.

GAN	Discriminator Loss	Generator Loss
MM GAN	$\mathcal{L}_{\mathrm{D}}^{\mathrm{GAN}} = -\mathbb{E}_{x\sim p_d}[\log(D(x))] - \mathbb{E}_{\hat{x}\sim p_g}[\log(1-D(\hat{x}))]$	$\mathcal{L}_{\mathrm{G}}^{\mathrm{GAN}} = \mathbb{E}_{\hat{x}\sim p_g}[\log(1-D(\hat{x}))]$
NS GAN	$\mathcal{L}_{\mathrm{D}}^{\mathrm{NSGAN}} = -\mathbb{E}_{x\sim p_d}[\log(D(x))] - \mathbb{E}_{\hat{x}\sim p_g}[\log(1-D(\hat{x}))]$	$\mathcal{L}_{\mathrm{G}}^{\mathrm{NSGAN}} = -\mathbb{E}_{\hat{x}\sim p_g}[\log(D(\hat{x}))]$
WGAN	$\mathcal{L}_{\mathrm{D}}^{\mathrm{WGAN}} = -\mathbb{E}_{x\sim p_d}[D(x)] + \mathbb{E}_{\hat{x}\sim p_g}[D(\hat{x})]$	$\mathcal{L}_{\mathrm{G}}^{\mathrm{WGAN}} = -\mathbb{E}_{\hat{x}\sim p_g}[D(\hat{x})]$
WGAN GP	$\mathcal{L}_{\mathrm{D}}^{\mathrm{WGANGP}} = \mathcal{L}_{\mathrm{D}}^{\mathrm{WGAN}} + \lambda\mathbb{E}_{\hat{x}\sim p_g}[(\|\nabla D(\alpha x + (1-\alpha\hat{x})\|_2 - 1)^2]$	$\mathcal{L}_{\mathrm{G}}^{\mathrm{WGANGP}} = -\mathbb{E}_{\hat{x}\sim p_g}[D(\hat{x})]$
LS GAN	$\mathcal{L}_{\mathrm{D}}^{\mathrm{LSGAN}} = -\mathbb{E}_{x\sim p_d}[(D(x)-1)^2] + \mathbb{E}_{\hat{x}\sim p_g}[D(\hat{x})^2]$	$\mathcal{L}_{\mathrm{G}}^{\mathrm{LSGAN}} = -\mathbb{E}_{\hat{x}\sim p_g}[(D(\hat{x}-1))^2]$
DRAGAN	$\mathcal{L}_{\mathrm{D}}^{\mathrm{DRAGAN}} = \mathcal{L}_{\mathrm{D}}^{\mathrm{GAN}} + \lambda\mathbb{E}_{\hat{x}\sim p_d+\mathcal{N}(0,c)}[(\|\nabla D(\hat{x})\|_2 - 1)^2]$	$\mathcal{L}_{\mathrm{G}}^{\mathrm{DRAGAN}} = \mathbb{E}_{\hat{x}\sim p_g}[\log(1-D(\hat{x}))]$
BEGAN	$\mathcal{L}_{\mathrm{D}}^{\mathrm{BEGAN}} = \mathbb{E}_{x\sim p_d}[\|x-\mathrm{AE}(x)\|_1] - k_t\mathbb{E}_{\hat{x}\sim p_g}[\|\hat{x}-\mathrm{AE}(\hat{x})\|_1]$	$\mathcal{L}_{\mathrm{G}}^{\mathrm{BEGAN}} = \mathbb{E}_{\hat{x}\sim p_g}[\|\hat{x}-\mathrm{AE}(\hat{x})\|_1]$

Figure: Models studied in Lucic et al. 2018

Ethics: Generative AI

Deepfakes:

- Initial model only applies autoencoder to swap faces.
  - Train: $\tilde{A} \to E \to h_{\tilde{A}} \to D_A \to A$,
  - Train: $\tilde{B} \to E \to h_{\tilde{B}} \to D_B \to B$
  - Generate: $A \to E \to h_A \to D_B \to B$
- Currently all faking technologies that apply deep neural networks.
- Becoming ever harder to detect.
- Impact on public trust in media.
- Popular use cases: face swapping, fake news

Figure: Deepfakes demo from `https://www.youtube.com/watch?v=QhxTTshL3b0`

Ethics: Generative AI

https://www.youtube.com/watch?v=cQ54GDm1eL0

Thank you very much!

yinchong.yang@siemens.com

Xi Chen et al. "Infogan: Interpretable representation learning by information maximizing generative adversarial nets". In: *Advances in neural information processing systems*. 2016, pp. 2172–2180.

William Fedus et al. "Many paths to equilibrium: GANs do not need to decrease a divergence at every step". In: *arXiv preprint arXiv:1710.08446* (2017).

Ian Goodfellow et al. "Generative adversarial nets". In: *Advances in neural information processing systems*. 2014, pp. 2672–2680.

Diederik P Kingma and Max Welling. "Auto-encoding variational bayes". In: *arXiv preprint arXiv:1312.6114* (2013).

Mario Lucic et al. "Are gans created equal? a large-scale study". In: *Advances in neural information processing systems*. 2018, pp. 700–709.

Mehdi Mirza and Simon Osindero. "Conditional generative adversarial nets". In: *arXiv preprint arXiv:1411.1784* (2014).

Jürgen Schmidhuber. "Learning factorial codes by predictability minimization". In: *Neural Computation* 4.6 (1992), pp. 863–879.

Jun-Yan Zhu et al. "Unpaired image-to-image translation using cycle-consistent adversarial networks". In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 2223–2232.