Generative Adversarial Networks

CMU 11-785 Introduction to Deep Learning, Fall 2020
lecture 23

Presenter : Changdae Oh
bnormal16@naver.com

TAVE Research DL001

2021.05.23

Topics

1. Generative Modeling
2. GANs and VAEs
3. GAN Theory
4. GAN Evaluation

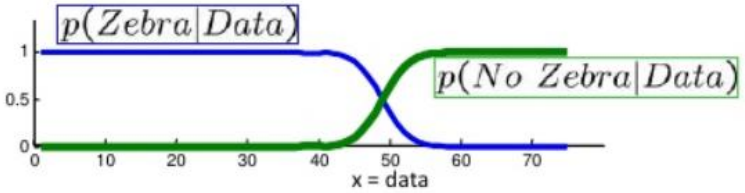
Topics

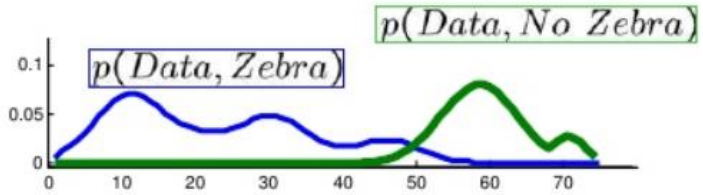
1. Generative Modeling
2. GANs and VAEs
3. GAN Theory
4. GAN Evaluation

Generative Modeling

Recap

	Discriminative model	Generative model
Training	Conditional prob. Estimation • $P_\theta(y \mid x)$ Learn direct maps • $y = f_\theta(x)$	Density estimation • $P_\theta(x)$ • $\boldsymbol{P_\theta(x, y)}$ or $P_\theta(x \mid y)$
Use in	Prediction $f : x \rightarrow \hat{y}$	Data generation • $g : seed \rightarrow \tilde{x}$ • $g : seed, y \rightarrow \tilde{x}$





Why Generative Networks?

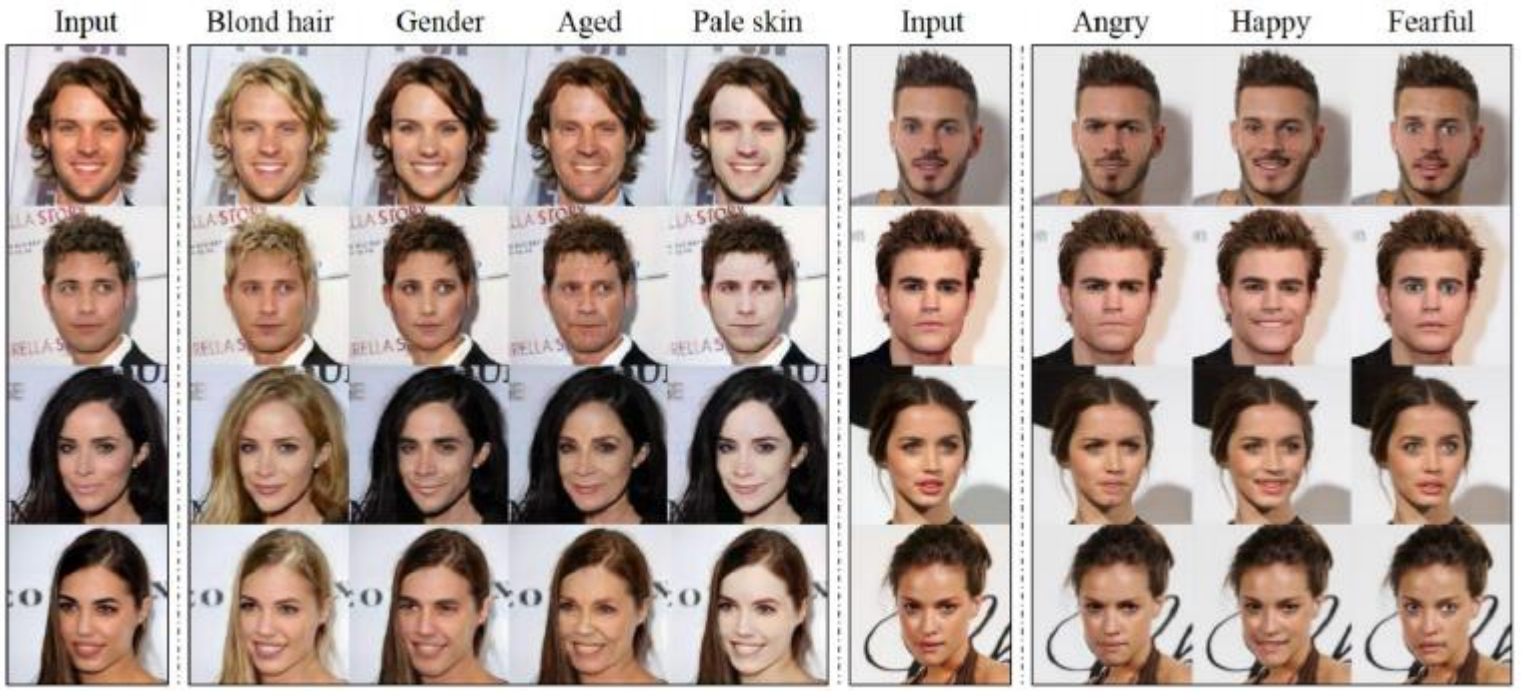


Figure 1. Multi-domain image-to-image translation results on the CelebA dataset via transferring knowledge learned from the RaFD dataset. The first and sixth columns show input images while the remaining columns are images generated by StarGAN. Note that the images are generated by a single generator network, and facial expression labels such as angry, happy, and fearful are from RaFD, not CelebA.

when you have a generative model,

that means you understand

what the data is structured like.

Beyond simply generating, it can be modified to a different aspects.

Generative Modeling

Why Generative Networks?

- Model understands the joint distribution P(X, Y)
  - Deeper understanding of the distribution than a discriminative model.
  - Better generalization can be achieved. (by extracting deep latent features.)
    - So, it can work well in discriminative task.
- However, model for P(X, Y) is harder to learn than P(Y | X)

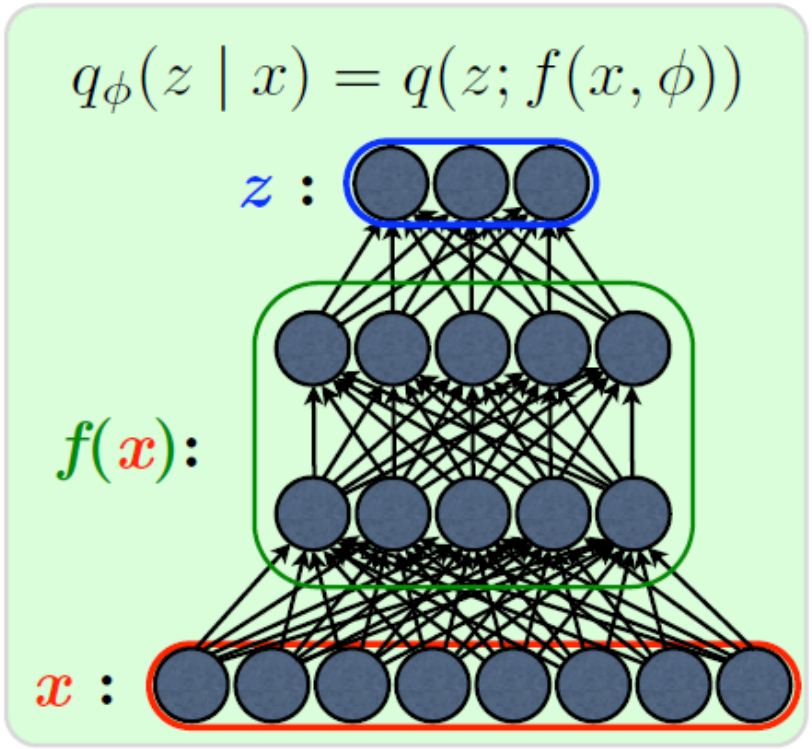
Implicit vs Explicit distribution modeling

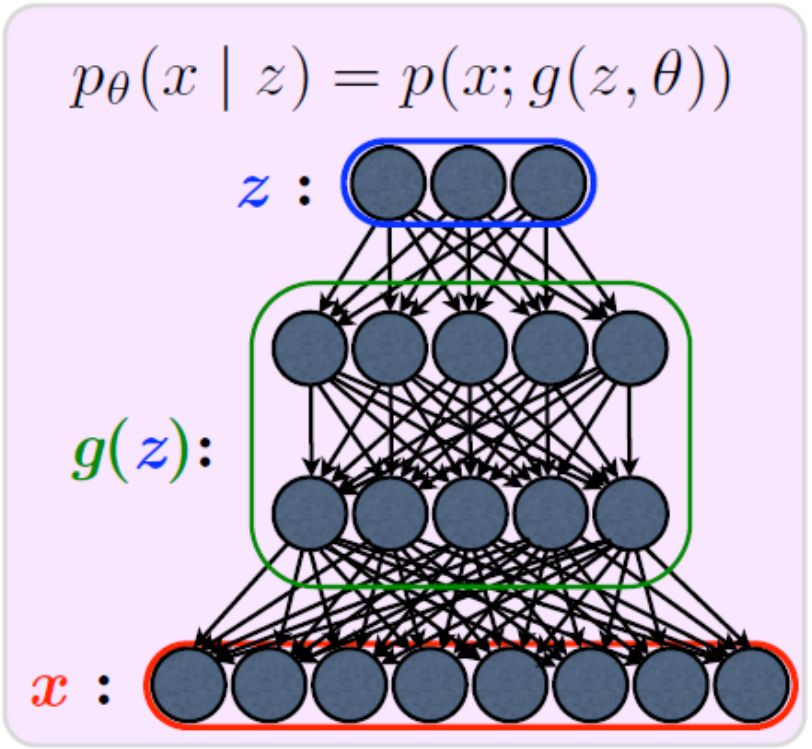
- **Explicit** : calculate P(x ~ X) for all x
  - Can capture the data directly.
  - The calculation is more difficult.
- **Implicit** : can generate samples x ~ X
  - Cannot tell the likelihood for a given data,
    but it can generate a reasonable data samples from its implicit distribution.
  - Easier to learn

Topics

1. Generative Modeling
2. GANs and VAEs
3. GAN Theory
4. GAN Evaluation

Review : VAE





- Encoder models Q(Z | X) ≈ P(Z | X), Decoder models P(X | Z)
- Trained to reconstruct inputs
  and encouraged the distribution over Z | X to match a prior P(Z)
- Hidden representation Z is learned by the model.

GANs and VAEs

GANs

- Minimize the divergence between the generated dist. and the target dist.
- Optimizing an estimate using sampling.
- Optimization is noisy and complicated but the model is attempting to model P(X) directly.
- Produce 'sharper' results.
- Only require the ability to sample from a prior.

VAEs

- Minimize a bound on the divergence between the generated dist. and the target dist.
- Optimizing a bound.
- Optimization is relatively straight forward but it isn't really optimizing P(X) and will get artifacts.
- Train faster and more reliably.
- Require an analytical understanding of the prior and its KL divergence.

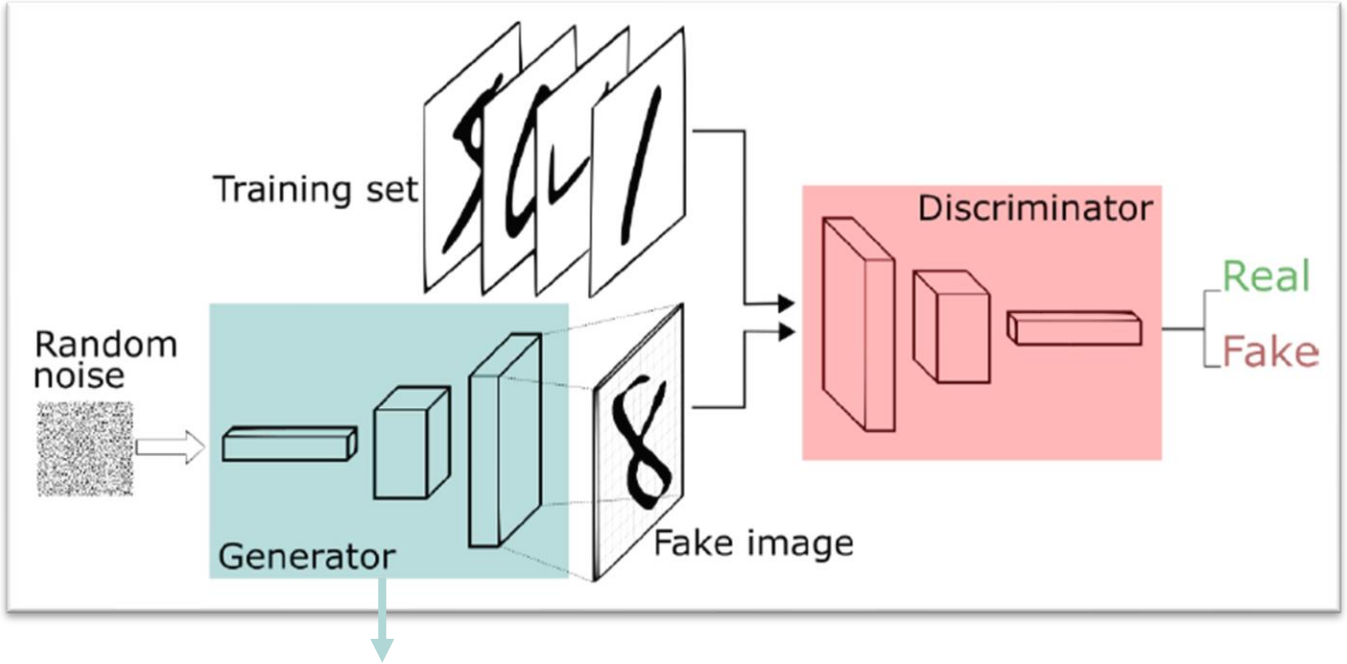
- VAEs learn an encoder-decoder pair but GANs do not.
- VAEs are more theoretically justified, the GANs more based on what works
- VAE generator trained on encoded data but evaluated on prior samples, GAN trained and evaluated on prior samples.

Topics

1. Generative Modeling
2. GANs and VAEs
3. GAN Theory
4. GAN Evaluation

Adversarial Learning

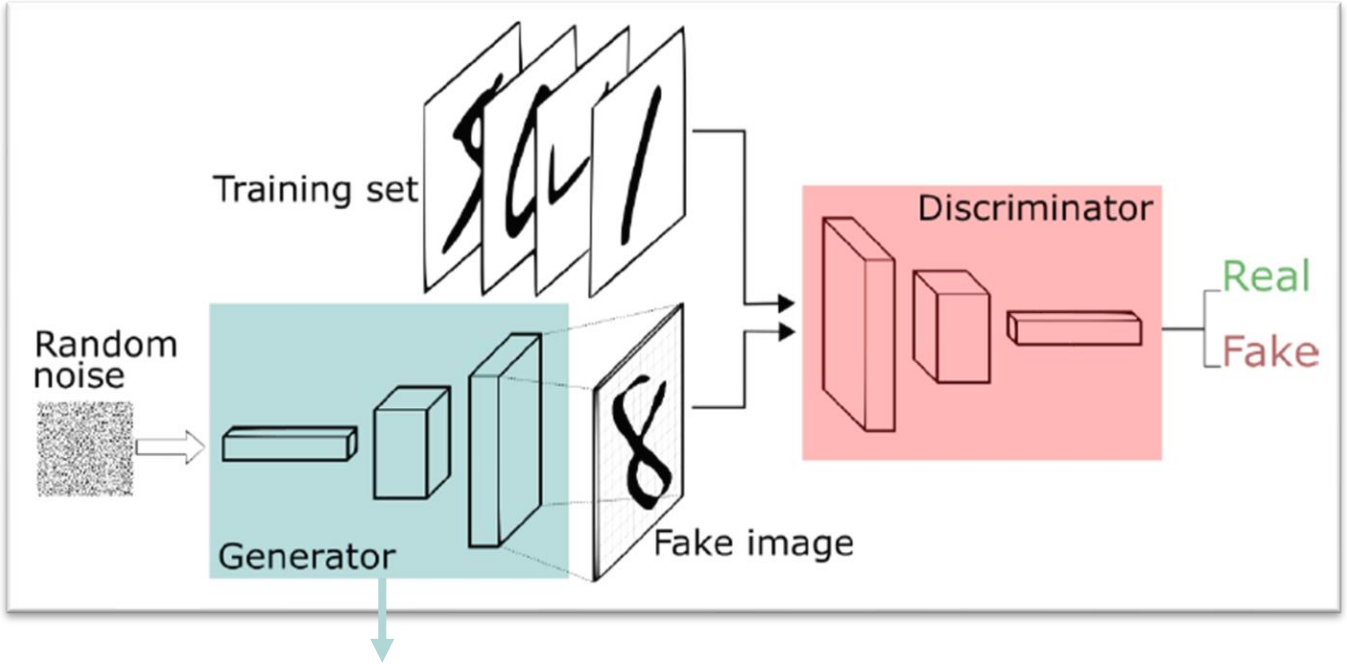
Goal : model P(X), the distribution of the training data.



- The generator learns P(X | Z)
  - Hidden representation Z is sampled from a known prior
  - Generator maps between a simple known distribution and a complicated output distribution.
    (learns a lower-dimensional manifold in the output space)
  - No simple loss function available to measure the divergence between the generated dist. and the real dist.
  - Instead of a traditional loss function, **loss is calculated by a discriminator.**

Adversarial Learning

Goal : model P(X), the distribution of the training data.



- The generator learns P(X | Z)

Goal of the generator is
for the generated distribution G(z) ~ Z to match the true P(X).

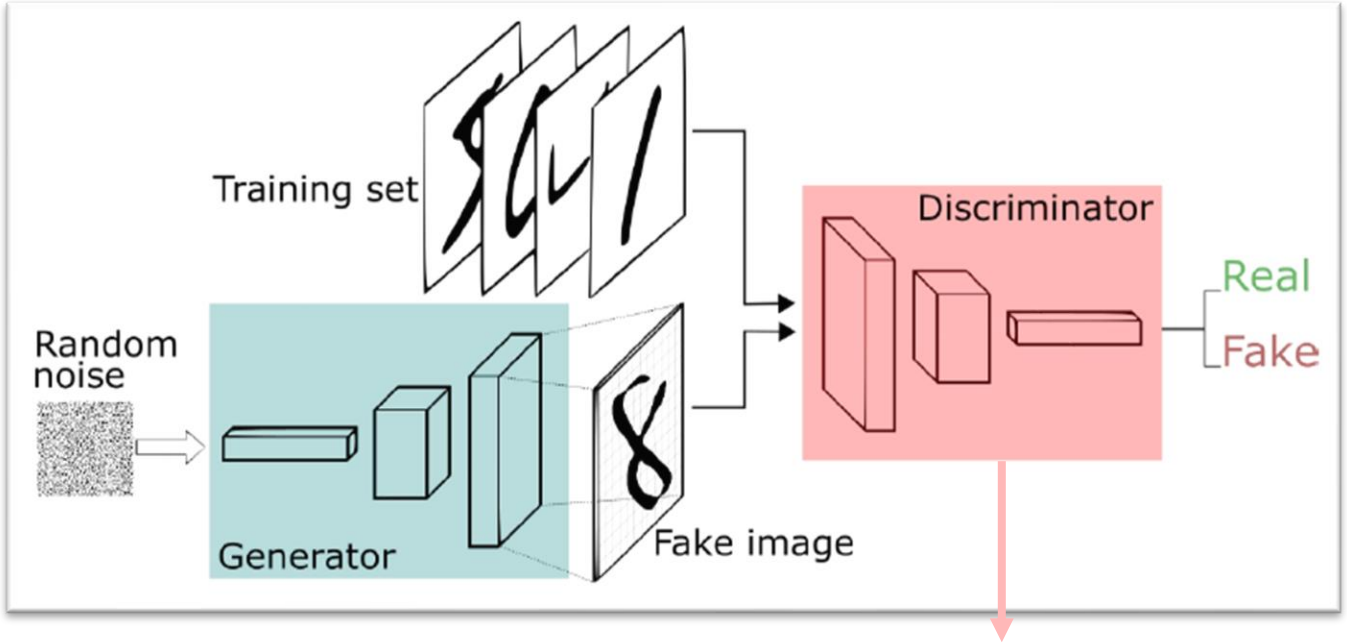
Sampling from some simple distribution Z,
put it into G, and get samples from P(X).

bution.

st. and the real dist.

Adversarial Learning

Goal : model P(X), the distribution of the training data.



- The discriminator is a secondary neural network that guides the generator
  - Trained to tell the difference between real and generated data
  - Generator tries to 'confuse' the discriminator, so it can't tell the difference between real and generated data
  - Discriminator tells generator how to look more 'real' and less 'fake' / 'generated'
  - **Serves the purpose of a 'loss function' in other models**

Math in GANs

Objective $\min_G \max_D V(D,G) = \mathbb{E}_X \log D(X) + \mathbb{E}_Z \log(1 - D(G(Z)))$

$$CE = -\sum_i^C t_i log(f(s)_i)$$

- The discriminator wants D(X) = 1 and D(G(Z)) = 0
- The generator wants D(G(Z)) = 1

The optimal discriminator

$$f := \mathbb{E}_{X\sim P_D} \log D(X) + \mathbb{E}_{X\sim P_G} \log(1 - D(X))$$
$$= \int_X [P_D(X) \log D(X) + P_G(X) \log(1 - D(X))]\, dX$$

$$\frac{\partial f}{\partial D(X)} = \frac{P_D(X)}{D(X)} - \frac{P_G(X)}{1 - D(X)} = 0$$
$$\frac{P_D(X)}{D(X)} = \frac{P_G(X)}{1 - D(X)}$$

$$D_{opt}(X) = \frac{P_D(X)}{P_G(X) + P_D(X)}$$

$$\mathbb{E}_{P_D} \log \frac{P_D(X)}{P_G(X) + P_D(X)} + \mathbb{E}_{P_G} \log \frac{P_G(X)}{P_G(X) + P_D(X)} \Rightarrow JSD(P_D|P_G) - \log 4$$

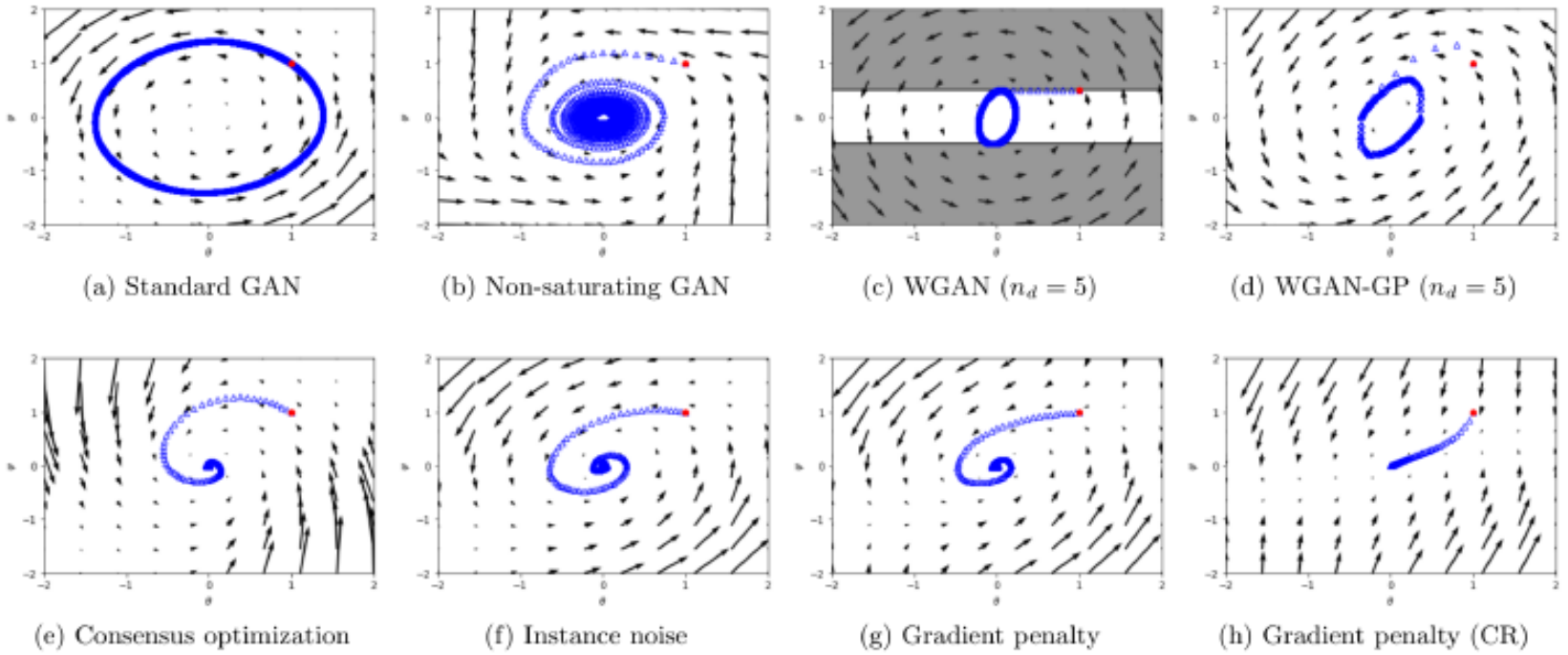
The optimal generator

$$\min_G JSD(P_D \| P_G) - \log 4$$

$$JSD(p\|q) = \frac{1}{2}KL(p\|M) + \frac{1}{2}KL(q\|M)$$
$$where, M = \frac{1}{2}(p+q)$$

Stationary Point & Stability

- The gradient to the generator is flat => no change in G dist.



- Stability : Are the arrows pointing towards the center?
- Stationary point : Presence of the center.

Difficulties in optimizing GANs

- Both generator and discriminator need to be trained simultaneously
- If the discriminator is under/over – trained ??
- The correct discriminator changes during training
- Discriminator and generator are trying to hit 'moving targets'

Perceptual Loss

- A discriminator might be able to address the ethereal issue of 'perceptual distance'
- Discriminator loss is much more flexible than L1, L2, etc.

Implicit Distributions

- A generator implicitly learns a target distribution P(X)
- Can draw samples from P(X) by drawing samples from P(Z) and calculating P(X | Z)

Topics

1. Generative Modeling
2. GANs and VAEs
3. GAN Theory
4. GAN Evaluation

GAN Evaluation

- The task of generating realistic-looking images is not easily quantified.
- The distribution is implicit and we cannot easily evaluate by something like calculating the likelihood of a test set.

Human eval

- The most direct answer to the question of whether generated data is 'realistic-looking'
- Expensive
- Time consuming
- Not reproducible
- subjective

Approx. likelihood

- Approximate the likelihood of a test set.
- Pull many samples of Z and calculate P(X|Z) for each, and then calculate the AVG prob
- Generate a million images, and count how many of those match your test point
- No image matches exactly, so generate a million images and place a Gaussian around each one and calculate the probably under GMM.
- Requires many samples, and some assumptions

Discriminative net

- A standard discriminative network can be trained to classify real images into some number of labels
- If the GAN is generating images correctly, disc model should produce a wide variety of labels. Each label should have high confidence

Code Review

https://github.com/changdaeoh/generative_model/tree/main/implementations

Code Review

```
class Sampling(keras.layers.Layer):
    def call(self, inputs):
        # sigma 대신 log(sigma^2)를 사용하는 변형.
        mean, log_var = inputs
        # 전달받은 평균, 분산을 통해 정의되는 정규분포 샘플을 리턴 (by reparameterization)
        return K.random_normal(tf.shape(log_var)) * K.exp(log_var / 2) + mean
```

```
# ------------------------------------------
# encoder
# ------------------------------------------
# mapping x to z
inputs = keras.layers.Input(shape=[28, 28])
z = keras.layers.Flatten()(inputs)
z = keras.layers.Dense(150, activation="selu")(z)
z = keras.layers.Dense(100, activation="selu")(z)
# learn mean & variance
codings_mean = keras.layers.Dense(codings_size)(z)
codings_log_var = keras.layers.Dense(codings_size)(z)
# sampling from the learned distribution of Z
codings = Sampling()([codings_mean, codings_log_var])
variational_encoder = keras.models.Model(
    inputs=[inputs], outputs=[codings_mean, codings_log_var, codings]
```

```
# ------------------------------------------
# decoder
# ------------------------------------------
# mapping z to x
decoder_inputs = keras.layers.Input(shape=[codings_size])
x = keras.layers.Dense(100, activation="selu")(decoder_inputs)
x = keras.layers.Dense(150, activation="selu")(x)
x = keras.layers.Dense(28 * 28, activation="sigmoid")(x)
outputs = keras.layers.Reshape([28, 28])(x)
variational_decoder = keras.models.Model(inputs=[docoder_inputs], outputs=[outputs])
```

```
# ------------------------------------------
# combined model : VAE
# ------------------------------------------
_, _, codings = variational_encoder(inputs)      # x to z
reconstructions = variational_decoder(codings)   # z to x hat
variational_ae = keras.models.Model(inputs=[inputs], outputs=[reconstructions])
# loss
latent_loss = -0.5 * K.sum(
    1 + codings_log_var - K.exp(codings_log_var) - K.square(codings_mean),
    axis=-1)

variational_ae.add_loss(K.mean(latent_loss) / 784.)
variational_ae.compile(loss="binary_crossentropy", optimizer="rmsprop", metrics=[rounded_accuracy])
history = variational_ae.fit(X_train, X_train, epochs=25, batch_size=128,
                             validation_data=(X_valid, X_valid))
```

Code Review

```
# ----------------------------------------
# model
# ----------------------------------------
codings_size = 30
# ---------------------------------------- generator
generator = keras.models.Sequential([
    keras.layers.Dense(100, activation="selu", input_shape=[codings_size]),
    keras.layers.Dense(150, activation="selu"),
    keras.layers.Dense(28 * 28, activation="sigmoid"),
    keras.layers.Reshape([28, 28])
])
# ---------------------------------------- discriminator
discriminator = keras.models.Sequential([
    keras.layers.Flatten(input_shape=[28, 28]),
    keras.layers.Dense(150, activation="selu"),
    keras.layers.Dense(100, activation="selu"),
    keras.layers.Dense(1, activation="sigmoid")
])
gan = keras.models.Sequential([generator, discriminator])
```

```
# ----------------------------------------
# train
# ----------------------------------------
discriminator.compile(loss="binary_crossentropy", optimizer="rmsprop")
discriminator.trainable = False
gan.compile(loss="binary_crossentropy", optimizer="rmsprop")

batch_size = 32
dataset = tf.data.Dataset.from_tensor_slices(X_train).shuffle(1000)
dataset = dataset.batch(batch_size, drop_remainder=True).prefetch(1)

def train_gan(gan, dataset, batch_size, codings_size, n_epochs=50):
    generator, discriminator = gan.layers
    for epoch in range(n_epochs):
        print("Epoch {}/{}".format(epoch + 1, n_epochs))
        for X_batch in dataset:
            # ------------------------- 1. training the discriminator
            noise = tf.random.normal(shape=[batch_size, codings_size]) # sampling Z
            generated_images = generator(noise)
            X_fake_and_real = tf.concat([generated_images, X_batch], axis=0) # concatenated data
            y1 = tf.constant([[0.]] * batch_size + [[1.]] * batch_size) # labels : fake & real
            discriminator.trainable = True
            discriminator.train_on_batch(X_fake_and_real, y1)
            # ------------------------- 2. training the generator
            noise = tf.random.normal(shape=[batch_size, codings_size])
            y2 = tf.constant([[1.]] * batch_size)
            discriminator.trainable = False # freezing discriminator
            gan.train_on_batch(noise, y2)

train_gan(gan, dataset, batch_size, codings_size, n_epochs=100)
```