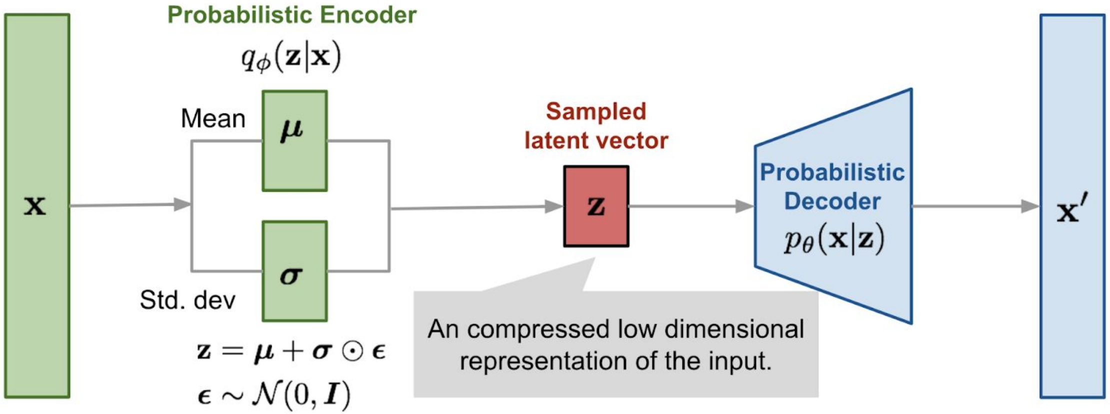
Generative Models

PHYS591000 2022.05.11

Review: VAE

- Last week we've learned that VAE is a kind of generative models, which learns how to generate (new) samples by estimating the probabilistic distribution of input (x) in the latent space (z)



Review: VAE

- And the math is complicated….

$$logP(x) = \int_z q(z|x)logP(x)dz$$

q(z|x) can be any distribution

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

$$= \int_z q(z|x)log\left(\frac{P(z,x)}{q(z|x)}\right)dz + \underbrace{\int_z q(z|x)log\left(\frac{q(z|x)}{P(z|x)}\right)dz}_{KL(q(z|x)||P(z|x))}$$

≥ 0

$$\geq \int_z q(z|x)log\left(\frac{P(x|z)P(z)}{q(z|x)}\right)dz$$

lower bound L_b

Lecture from Prof. Hung-Yi Lee (NTU)

Motivation of GAN

- What if we just learn the ability of sampling (from a distribution) and generating samples, but don’t need to explicitly model the distribution?

  → Generative Adversarial Network (GAN)

Ref:
Lecture 18 by Prof. Hung-Yi Lee (youtube)
Lecture 13 of CS231(2017) at Stanford (youtube)

GAN

Ian Goodfellow et al., "Generative Adversarial Nets", NIPS 2014

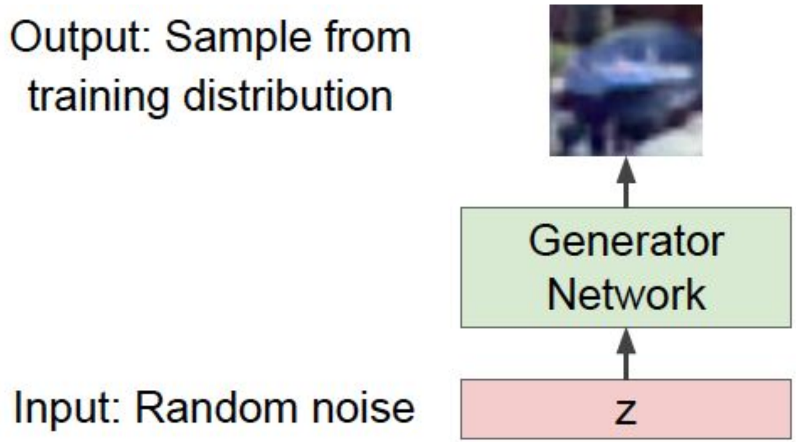
- Goal: Sampling from a random (Gaussian) distribution and generate a sample.

- Need: A complicated mapping from random noises to underlying P(x|z) (without expliciting modeling P(x|z)).

GAN

Ian Goodfellow et al., "Generative Adversarial Nets", NIPS 2014

- Solution: Use a NN (**Generator**) for the mapping



Lecture 13 of Stanford CS231 (2017)

- Q: How to train the Generator?

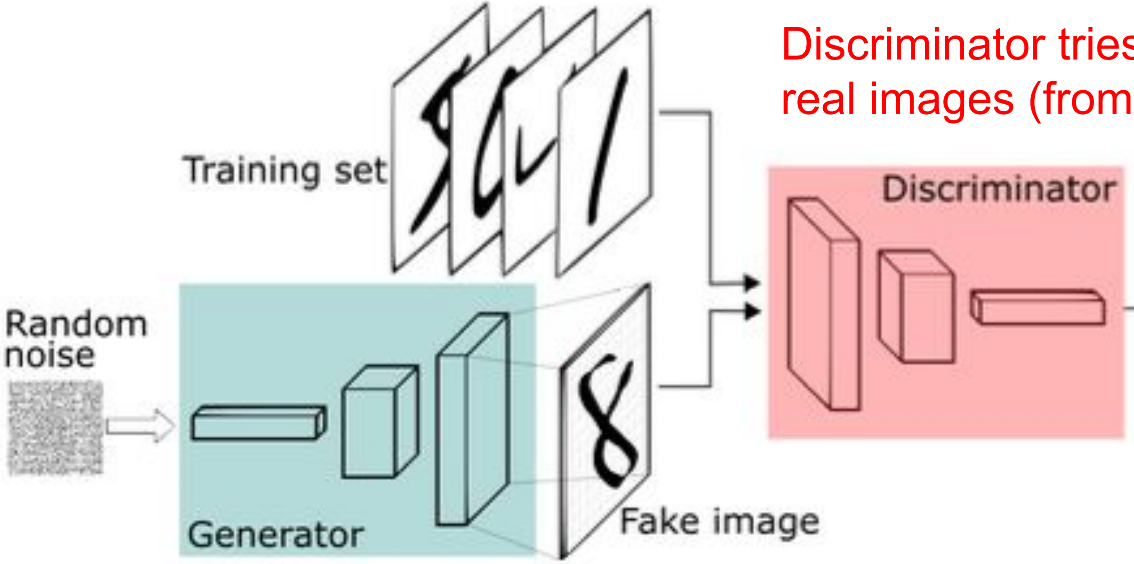
GAN

- Idea: A two-player game between a **generator** and a **discriminator**:
  - Discriminator (binary classifier): Tries to distinguish real data from data generated by the generator (fake data).
  - Generator: Tries to generate fake data that can fool the discriminator by tuning its parameters

GAN

Discriminator tries to identify the real images (from training data)

Input



1

0

Output

Generator tries to generate fake images that can gets high scores (close to 1) from the discriminator.

GAN+CNN = DCGAN

- Recall that CNN is a powerful tool for image processing → Can use a convolutional network for the discriminator, and a ‘deconvolutional’ network* for the generator.

- This is the idea of Deep Convolutional GAN (DCGAN), which makes uses of convolutional layers w/o pooling or FC layers.

* Can be implemented with UpSampling2D or Conv2DTranspose layers in Keras.

Training GAN

- To train a GAN means training two networks (generator + discriminator) together:
  1. Train the discriminator for k (k>=1) times with fake images produced by the generator and true images from training data.
  2. *Fix the (parameters of the) discriminator* and train the generator to produce images that gets a high score (‘looked real’) from the discriminator.
  3. Repeat 1 and 2.

In-class demo for this week

- Again we use MNIST dataset to illustrate the concept of GAN
- Feel free to play with it afterwards. We highlight a few points below.

```python
latent_size = 100
img_shape = (28,28,1)

# Use CNN, i.e. DCGAN
# Discriminator
# input = image, output = binary classifier
discriminator = Sequential()
discriminator.add(Conv2D(64, (3,3), strides=(2, 2), padding='same', inp
ut_shape=img_shape))
discriminator.add(LeakyReLU(alpha=0.2))
discriminator.add(Dropout(0.4))
discriminator.add(Conv2D(64, (3,3), strides=(2, 2), padding='same'))
discriminator.add(LeakyReLU(alpha=0.2))
discriminator.add(Dropout(0.4))
discriminator.add(Flatten())
discriminator.add(Dense(1, activation='sigmoid'))
discriminator.compile(loss='binary_crossentropy',
                      optimizer=Adam(0.0002, 0.5),
                      metrics=['accuracy'])
```

The standalone Discriminator part includes the compile part (how to update the parameters)

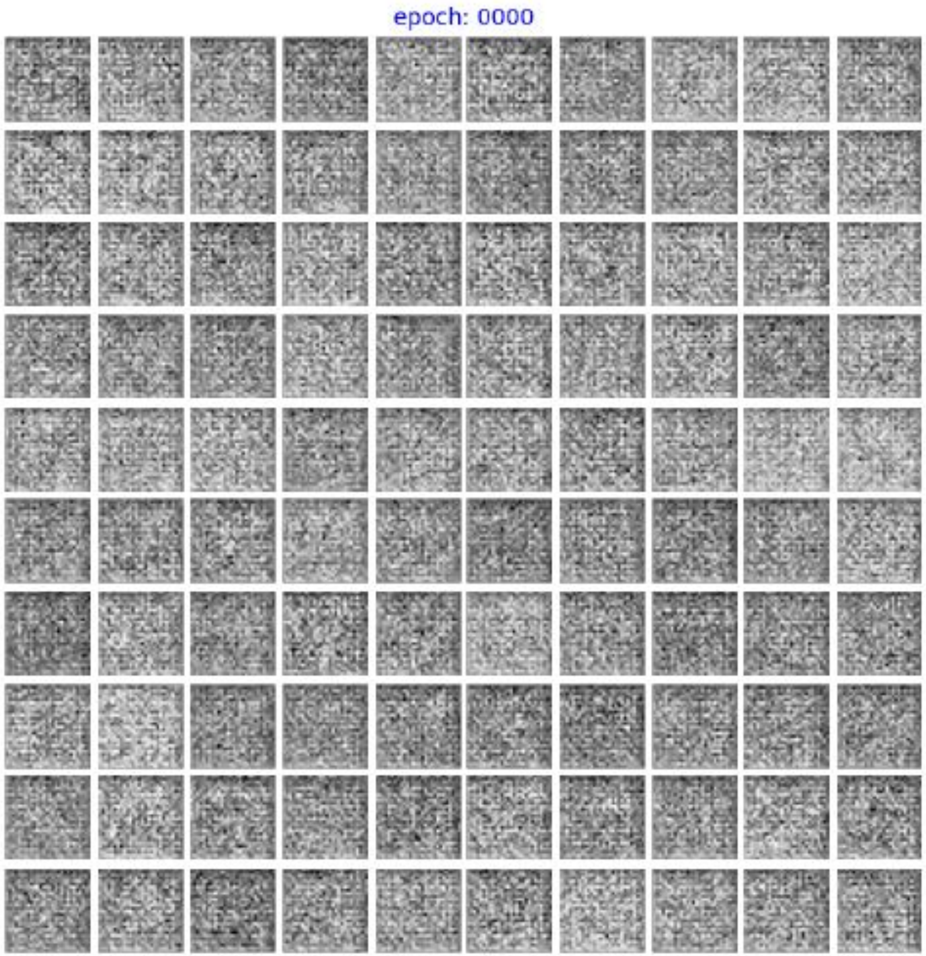
The standalone Generator part does not include the compile part. Its parameter will be updated with the Discriminator fixed.

```
generator = Sequential()
n_nodes = 128*7*7
# Start with 7x7 image
generator.add(Dense(n_nodes, input_dim=latent_size))
generator.add(LeakyReLU(alpha=0.2))
generator.add(Reshape((7, 7, 128)))
# Upsample to 14x14
generator.add(Conv2DTranspose(128, (4,4), strides=(2,2), padding='same
'))
generator.add(LeakyReLU(alpha=0.2))
# Upsample to 28x28
generator.add(Conv2DTranspose(128, (4,4), strides=(2,2), padding='same
'))
generator.add(LeakyReLU(alpha=0.2))
generator.add(Conv2D(1, (7,7), activation='softplus', padding='same'))
```

```python
def combined_gan(g,d):
    d.trainable = False
    model = Sequential()
    model.add(g)
    model.add(d)
    model.compile(loss='binary_crossentropy', optimizer=Adam(0.0002, 0.
5))
    return model
```

The standalone Generator part does not include the compile part. Its parameter will be updated with the Discriminator fixed.

→ The compile part is added when we combine the two NN into the full GAN.



First epoch: Begin with complete noise.

After 500 epochs: Start to see some structure.

After 1000 epochs: Slowly becomes legible.

Training GAN

- Training two networks together is in fact challenging:
  - Vanishing gradient
  - Mode collapse: Generator only learns to produce a particular subset of output which can easily fool the discriminator.
    (E.g. Produces only ‘perfect’ 0 and 1, but no other digits.)
  - Fail to converge (unstable)

Training GAN

- Improving the stability of GAN is an active research area!
  E.g. Wasserstein GAN (WGAN) makes use of a clever loss function for the discriminator to alleviate the vanishing gradient and mode collapse problems.

Applications of GAN

- Once we have trained a GAN successfully, we can use the generator to do many things:
  - Doodle → Photo (image-image translation)

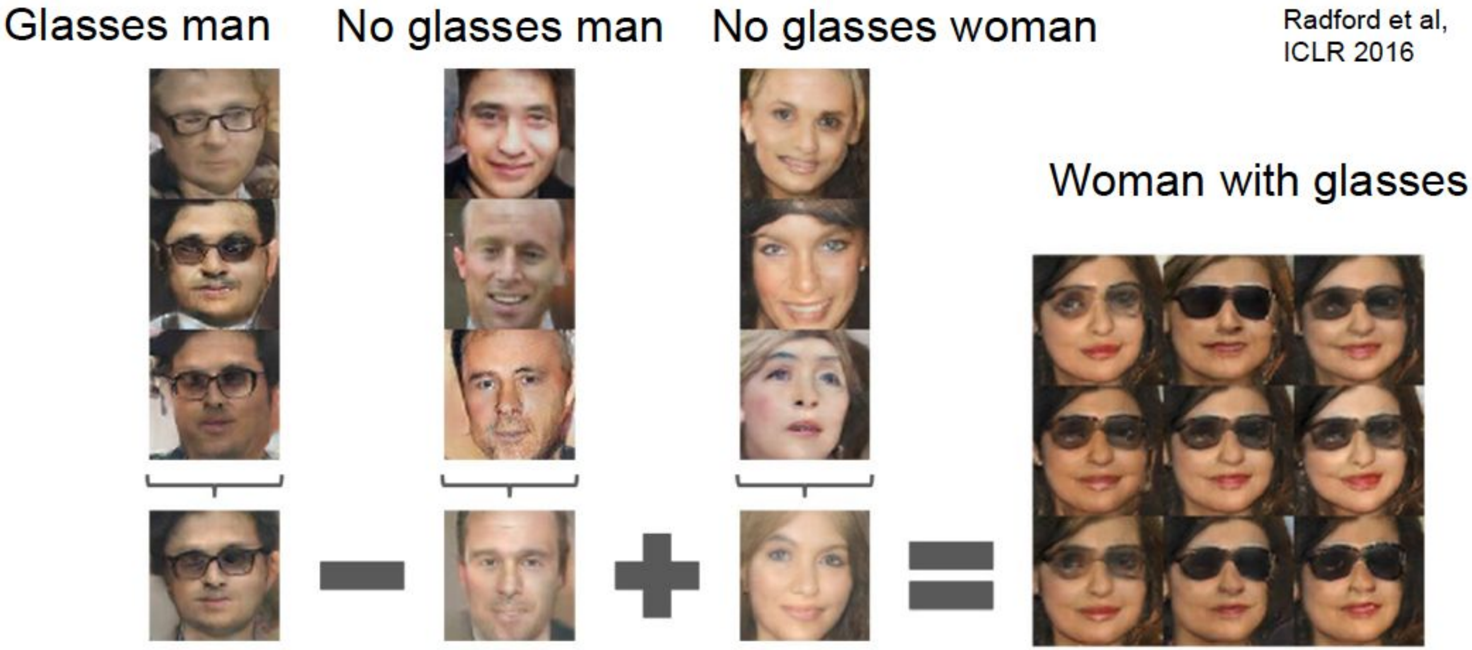
Applications of GAN

- Once we have trained a GAN successfully, we can use the generator to do many things:
  - Doodle → Photo
    (image-image translation)
  - CycleGAN
    Horse → Zebra
    (image transformation)

Applications of GAN

- Interpretable math:

Lecture 13 of Stanford CS231 (2017)



The GAN Zoo

Modifying the Optimization of GAN	Different Structure from the Original GAN
fGAN	Conditional GAN
WGAN	Semi-supervised GAN
Least-square GAN	InfoGAN
Loss Sensitive GAN	BiGAN
Energy-based GAN	Cycle GAN
Boundary-seeking GAN	Disco GAN
Unroll GAN	VAE-GAN
……	……

Lecture from Prof. Hung-Yi Lee (NTU)

In-class demo for this week

- You can check out the link below for examples of implementing various GAN in Keras and applying on MNIST: https://github.com/eriklindernoren/Keras-GAN

Lab for this week

- For Lab this week we’ll work with FlyCircuit data again, this time with GAN!

- This will be the last Lab assignment for this semester.
Due **next Wednesday (May 18) 5PM**.

Outlook for the semester

- Next week (May 18):
  – No Lab.
  – Remark on topics we don’t have time to cover.
  – Details on how to prepare for your final project presentations.

- Next next week (May 25): Guest Lecture by Prof. Daw-Wei Wang.

Backup