Frameworks de IA

Curitiba/2023

1

6 GAN – *Generative Adversarial Network*

2

Prof. Dr. Razer A. N. R. Montaño – UFPR/SEPT

Tensorflow

2

3

GAN

- Usadas basicamente para geração de dados (imagens)
- https://thispersondoesnotexist.com/
- http://www.whichfaceisreal.com/

- Fonte: https://www.tensorflow.org/tutorials/generative/dcgan

3

4

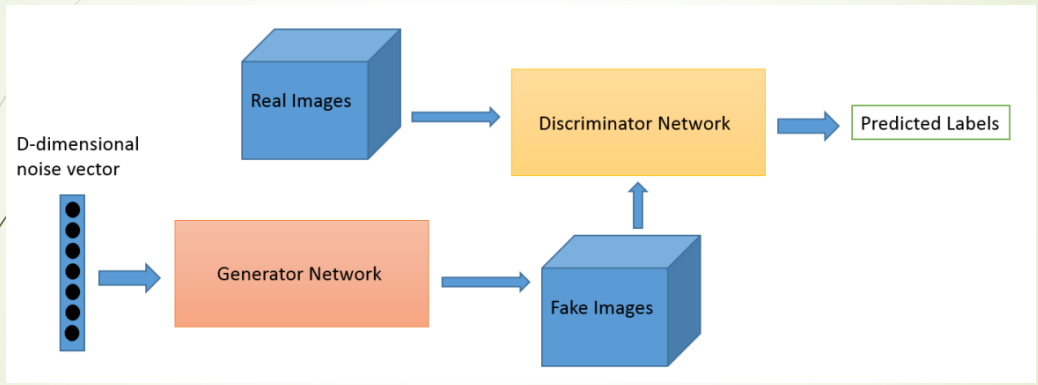
GAN - Arquitetura

- Formado por 2 redes neurais
  - **Gerador** : gera imagens parecidas com reais
  - **Discriminador** : verifica se são imagens reais ou *fake*
- **Gerador** e **Discriminador** são adversários e um tenta vencer o outro

4

5

GAN - Arquitetura

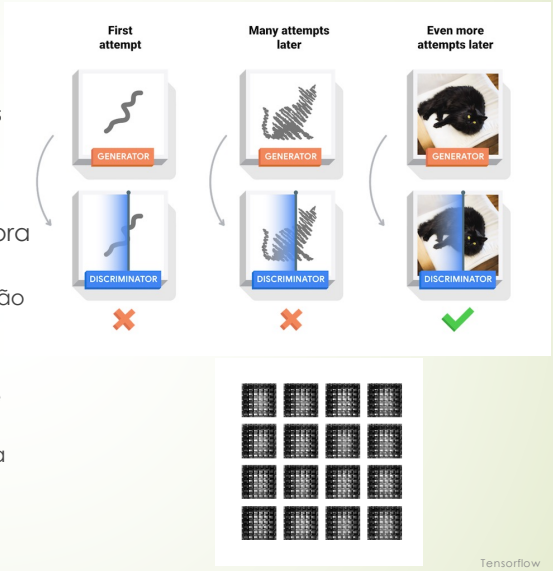


5

6

GAN - Arquitetura

- Processo
  - **Gerador** : no início não gera imagens muito boas
  - **Discriminador** : no início não detecta bem
- Mas conforme o Discriminador melhora a detecção
  - O Gerador precisa melhorar a geração de imagens
- Ciclo continua até que:
  - O Discriminador está bom o bastante para não ser enganado
  - O Gerador está bom o bastante para conseguir enganar

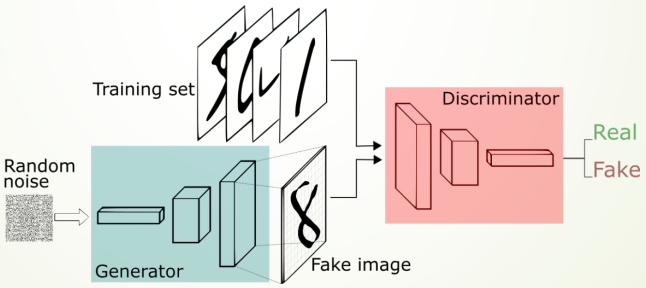


6

7

GAN - Arquitetura

- **Gerador** : É basicamente uma CNN invertida
  - Obtém um vetor aleatório unidimensional de ruído
  - Considera esse vetor como um mapa de atributos
  - Parte do mapa de atributos e gera a imagem
  - `Input` → `Dense` → `Conv2DTranspose`

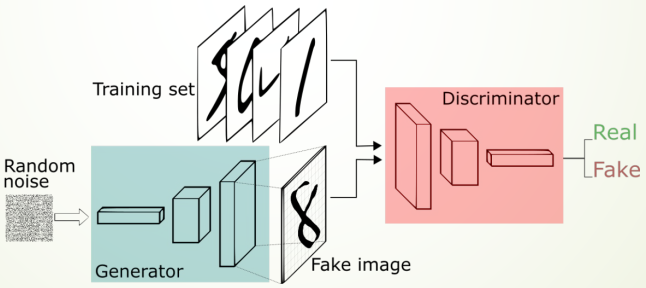


7

8

GAN - Arquitetura

- **Conv2DTranspose**
  - Camanda de convolução transposta (*upsampling*)
  - Conhecida como Deconvolução, sentido oposto da convolução
  - **Recebe**: dados com *shape* de saída de um convolução
  - **Retorna**: dados com *shape* de entrada de uma convolução



8

9

GAN – Arquitetura.

- Alguns detalhes
  - Função de perda é ***Binary Cross Entropy*** (2 classes)
  - Otimizadores podem ser diferentes, aqui será o **Adam** para os dois
- Duas funções:
  - **`discriminator_loss()`**: retorna o quão bom o discriminador distingue imagens fake de reais
  - **`generator_loss()`**: retorna o quão bom o gerador engana o discriminador

9

10

Laboratório

- Gerar novas imagens *fake* de dígitos a partir da base MNIST
- Usar GAN

10

11

Imagens Fake de Dígitos

- Importações

```
import tensorflow as tf

import glob
import imageio
import matplotlib.pyplot as plt
import numpy as np
import os
import PIL
from tensorflow.keras import layers
import time

from IPython import display
```


11

12

Imagens Fake de Dígitos

- Instalar dependências

```
!pip install imageio
```

```
!pip install git+https://github.com/tensorflow/docs
```


12

13

Imagens Fake de Dígitos

- Carregar os dados e pré-processar

```
# Carga, pré-processamento e separação

(train_images, train_labels), (_, _) = tf.keras.datasets.mnist.load_data()

train_images = train_images.reshape(train_images.shape[0], 28, 28, 1).astype('float32')
train_images = (train_images - 127.5) / 127.5  # Normalize the images to [-1, 1]

BUFFER_SIZE = 60000
BATCH_SIZE = 256

# Batch and shuffle the data
train_dataset =
tf.data.Dataset.from_tensor_slices(train_images).shuffle(BUFFER_SIZE).batch(BATCH_SIZE)
```


13

14

Imagens Fake de Dígitos

- Cria o gerador

```
def make_generator_model():
    model = tf.keras.Sequential()
    model.add(layers.Dense(7*7*256, use_bias=False, input_shape=(100,)))
    model.add(layers.BatchNormalization())
    model.add(layers.LeakyReLU())

    model.add(layers.Reshape((7, 7, 256)))
    assert model.output_shape == (None, 7, 7, 256)  # Note: None is the batch size

    model.add(layers.Conv2DTranspose(128, (5, 5), strides=(1, 1), padding='same', use_bias=False))
    assert model.output_shape == (None, 7, 7, 128)
    model.add(layers.BatchNormalization())
    model.add(layers.LeakyReLU())

    model.add(layers.Conv2DTranspose(64, (5, 5), strides=(2, 2), padding='same', use_bias=False))
    assert model.output_shape == (None, 14, 14, 64)
    model.add(layers.BatchNormalization())
    model.add(layers.LeakyReLU())

    model.add(layers.Conv2DTranspose(1, (5, 5), strides=(2, 2), padding='same', use_bias=False,
activation='tanh'))
    assert model.output_shape == (None, 28, 28, 1)

    return model
```

14

15

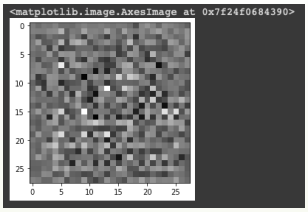
Imagens Fake de Dígitos

- Testa o gerador

```
# teste de criação de uma imagem (sem treinar)
generator = make_generator_model()

noise = tf.random.normal([1, 100])
generated_image = generator(noise, training=False)

plt.imshow(generated_image[0, :, :, 0], cmap='gray')
```



15

16

Imagens Fake de Dígitos

- Cria o discriminador

```
def make_discriminator_model():
    model = tf.keras.Sequential()
    model.add(layers.Conv2D(64, (5, 5), strides=(2, 2), padding='same',
                            input_shape=[28, 28, 1]))
    model.add(layers.LeakyReLU())
    model.add(layers.Dropout(0.3))

    model.add(layers.Conv2D(128, (5, 5), strides=(2, 2), padding='same'))
    model.add(layers.LeakyReLU())
    model.add(layers.Dropout(0.3))

    model.add(layers.Flatten())
    model.add(layers.Dense(1))

    return model
```


16

17

Imagens Fake de Dígitos

- Testa o discriminador

```
# Teste do discriminador (não treinado)
discriminator = make_discriminator_model()
decision = discriminator(generated_image)
print (decision)
```

```
tf.Tensor([[0.00153092]], shape=(1, 1), dtype=float32)
```


17

18

Imagens Fake de Dígitos

- Função de perda e otimizador

```
#Função de perda Binary Cross Entropy
cross_entropy = tf.keras.losses.BinaryCrossentropy(from_logits=True)

# Quantifica quão bom é o discriminador
# Compara: predições de imagens reais com vetor de 1's
# Compara: predições de imagens fake com vetor de 0's
def discriminator_loss(real_output, fake_output):
    real_loss = cross_entropy(tf.ones_like(real_output), real_output)
    fake_loss = cross_entropy(tf.zeros_like(fake_output), fake_output)
    total_loss = real_loss + fake_loss
    return total_loss

# Quantifica quão bem o gerador enganou o discriminador
# Se for bom, imagens fake serão classificadas com 1's
# Compara: predições fake com 1's
def generator_loss(fake_output):
    return cross_entropy(tf.ones_like(fake_output), fake_output)

# Otimizador do gerador e discriminador
generator_optimizer = tf.keras.optimizers.Adam(1e-4)
discriminator_optimizer = tf.keras.optimizers.Adam(1e-4)
```


18

19

Imagens Fake de Dígitos

- Salvar pontos de verificação

```
# Salvar pontos de verificação, caso a tarefa seja interrompida
checkpoint_dir = './training_checkpoints'
checkpoint_prefix = os.path.join(checkpoint_dir, "ckpt")
checkpoint = tf.train.Checkpoint(generator_optimizer=generator_optimizer,
                                 discriminator_optimizer=discriminator_optimizer,
                                 generator=generator,
                                 discriminator=discriminator)
```


19

20

Imagens Fake de Dígitos

- Loop de treinamento

```
EPOCHS = 50
noise_dim = 100
num_examples_to_generate = 16
seed = tf.random.normal([num_examples_to_generate, noise_dim])

@tf.function
def train_step(images):
    noise = tf.random.normal([BATCH_SIZE, noise_dim])

    with tf.GradientTape() as gen_tape, tf.GradientTape() as disc_tape:
      generated_images = generator(noise, training=True)

      real_output = discriminator(images, training=True)
      fake_output = discriminator(generated_images, training=True)

      gen_loss = generator_loss(fake_output)
      disc_loss = discriminator_loss(real_output, fake_output)

    gradients_of_generator = gen_tape.gradient(gen_loss, generator.trainable_variables)
    gradients_of_discriminator = disc_tape.gradient(disc_loss, discriminator.trainable_variables)

    generator_optimizer.apply_gradients(zip(gradients_of_generator, generator.trainable_variables))
    discriminator_optimizer.apply_gradients(zip(gradients_of_discriminator,
discriminator.trainable_variables))
```

20

21

Imagens Fake de Dígitos

- Loop de treinamento

```
def train(dataset, epochs):
  for epoch in range(epochs):
    start = time.time()

    for image_batch in dataset:
      train_step(image_batch)

    # Produz as imagens para o GIF
    display.clear_output(wait=True)
    generate_and_save_images(generator, epoch + 1, seed)

    # Salva a cada 15 épocas
    if (epoch + 1) % 15 == 0:
      checkpoint.save(file_prefix = checkpoint_prefix)

    print ('Time for epoch {} is {} sec'.format(epoch + 1, time.time()-start))

  # Gera depois da época final
  display.clear_output(wait=True)
  generate_and_save_images(generator, epochs, seed)
```


21

22

Imagens Fake de Dígitos

- Geração das imagens

```
# Gerar e salvar imagens
def generate_and_save_images(model, epoch, test_input):
  # Notice `training` is set to False.
  # This is so all layers run in inference mode (batchnorm).
  predictions = model(test_input, training=False)

  fig = plt.figure(figsize=(4, 4))

  for i in range(predictions.shape[0]):
      plt.subplot(4, 4, i+1)
      plt.imshow(predictions[i, :, :, 0] * 127.5 + 127.5, cmap='gray')
      plt.axis('off')

  plt.savefig('image_at_epoch_{:04d}.png'.format(epoch))
  plt.show()
```


22

23

Imagens Fake de Dígitos

- Treino do modelo

```
# Treinar o modelo

train(train_dataset, EPOCHS)

# restaurar o último ponto de verificação

checkpoint.restore(tf.train.latest_checkpoint(checkpoint_dir))
```


23

24

Imagens Fake de Dígitos.

- Gerar e mostrar o GIF animado

```
# Mostra uma imagem conforme a época
def display_image(epoch_no):
  return PIL.Image.open('image_at_epoch_{:04d}.png'.format(epoch_no))

display_image(EPOCHS)

# Gera e mostra o GIF
anim_file = 'dcgan.gif'

with imageio.get_writer(anim_file, mode='I') as writer:
  filenames = glob.glob('image*.png')
  filenames = sorted(filenames)
  for filename in filenames:
    image = imageio.imread(filename)
    writer.append_data(image)
  image = imageio.imread(filename)
  writer.append_data(image)

import tensorflow_docs.vis.embed as embed
embed.embed_file(anim_file)
```


24


25
EXERCÍCIO.
Executar o exercício de GANs.
DESAFIO:
Gerar novos rostos treinando com rostos de celebridades:
https://www.kaggle.com/jessicali9530/celeba-dataset
Prof. Dr. Razer A. N. R. Montaño – UFPR/SEPT
Tensorflow

25