Aula 8
Geração artificial de dados

Eduardo L. L. Cabral

Objetivos

- Apresentar geração artificial de dados (“data augmentation”).
- Apresentar como gerar dados artificialmente usando o TensorFlow-Keras.
- Formas de utilizar geração artificial de dados.
- Apresentar como treinar uma RNA com o TensorFlow-Keras usando um gerador com geração artificial de dados.

Motivação

- Existe um balanço entre “overfitting” e “underfitting” $\Rightarrow$ uma RNA com bom desenho possui muitos parâmetros e deve ser treinada com um conjunto muito grande de dados para evitar problemas de “overfitting”.
- RNAs que apresentam alto desempenho possuem muitos parâmetros e camadas para serem capazes de aproximar adequadamente funções complexas $\Rightarrow$ como é o caso de quase todos os problemas reais.
- Se uma RNA tem muitos parâmetros, precisa de uma quantidade proporcional de exemplos para obter um bom desempenho para dados novos.

Motivação

- Número de parâmetros em redes neurais populares VGGNet, DeepVideo e GNMT:

	VGGNet	DeepVideo	GNMT
Used For	Identifying Image Category	Identifying Video Category	Translation
Input	Image 	Video 	English Text 
Output	1000 Categories	47 Categories	French Text
Parameters	140M	~100M	380M
Data Size	1.2M Images with assigned Category	1.1M Videos with assigned Category	6M Sentence Pairs, 340M Words
Dataset	ILSVRC-2012	Sports-1M	WMT'14

https://nanonets.com/blog/data-augmentation-how-to-use-deep-learning-when-you-have-limited-data-part-2/

- Redes neurais de última geração normalmente têm milhões de parâmetros!

Motivação

- Como treinar uma RNA com milhões de parâmetros sem problema de “overfitting”?

  $\Rightarrow$ Obter mais dados é a solução mais eficiente.

- Mas como obter mais dados, se na maior parte dos casos não se tem dados disponíveis?

  $\Rightarrow$ A forma mais simples é obter mais dados usando geração artificial de dados.

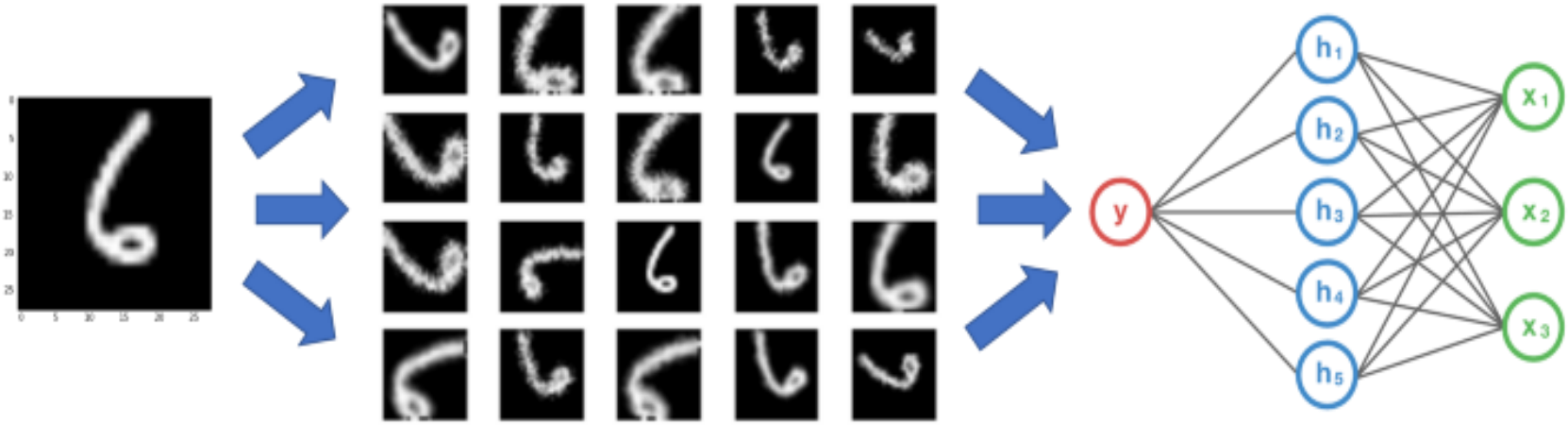
- “Data augmentation” é uma estratégia usada para aumentar a quantidade de dados usando técnicas como recorte, inversão, translação, rotação, inserção de ruído, alteração de tonalidades de cor etc.

Motivação

- A técnica de “data augmentation” é mais utilizada para imagens.
- Uma RNA convolucional capaz de classificar objetos de maneira robusta deve ter a propriedade de invariância à translação, rotação, ponto de vista, tamanho, iluminação etc (ou uma combinação desses itens).
- Em geral o conjunto de dados de imagens são obtidas em condições limitadas, porém a RNA deve ser capaz de identificar o mesmo objeto em diferentes condições de orientação, localização, escala, brilho etc.
- A obtenção de novos dados com muitas variações é a premissa básica da geração artificial de dados.

Motivação

- No processo de “data augmentation” são feitas pequenas alterações nos dados existentes:



⇒ Para uma RNA essas imagens são distintas.

Técnicas de “data augmentation”

- As técnicas de geração artificial de dados a partir de dados existentes são:
  1. Inversão;
  2. Rotação;
  3. Translação;
  4. Corte;
  5. Mudança de escala;
  6. Inserção de ruído;
  7. Distorção;
  8. Mudança de brilho;
  9. Cisalhamento;
  10. Etc.

Como fazer no TensorFlow-Keras

- Classe de gerador de dados `ImageDataGenerator` do Keras realiza “data augmentation” automaticamente.
- `ImageDataGenerator` gera lotes de imagens “augmentadas” em “tempo real” durante o treinamento.
- Imagens são geradas a partir das imagens originais e são descartadas logo após o seu uso $\Rightarrow$ não ocupam espaço em memória.
- Novas imagens criadas pelo gerador tem as mesmas dimensões das imagens originais.

Como fazer no TensorFlow-Keras

- Existem vários métodos que podem ser usados com a classe `ImageGenerator` do Keras.
- Na aula anterior vimos o método `flow_from_directory`, que recebe um caminho para o diretório onde estão as imagens.
- Outro método é o `flow`, que recebe um tensor de dados e gera um lote de novas imagens, criadas a partir desses dados conforme as operações de transformações escolhidas.
- Vamos usar o método `flow` para entender como realizar “data augmentation” com o Keras.
- Como exemplo das transformações realizadas vamos utilizar uma única imagem de entrada e gerar 4 novas imagens a partir dessa.

Como fazer no TensorFlow-Keras

- Primeiramente deve-se carregar a imagem que é usada como base para gerar as novas imagens.
- O código a seguir carrega e mostra a imagem.

```
# Importa bibliotecas
from skimage.io import imread
import numpy as np
import matplotlib.pyplot as plt

# Carrega imagem do arquivo,
# transforma em um tensor numpy
# e mostra
img = imread("casa.jpg")
img = np.array(img)
print(img.shape)
plt.imshow(img)
plt.show()
```

Como fazer no TensorFlow-Keras

- O Keras espera que o primeiro eixo dos dados de entrada seja o eixo dos exemplos, então temos que adicionar um eixo a mais à nossa imagem (eixo 0).

```
# Cria um conjunto de dados que contém somente uma imagem
imagens = img.reshape((1, img.shape[0], img.shape[1],
    img.shape[2]))

# Mostra nova dimensão da imagem original e da imagem com
eixo adicional
print('Dimensão imagem original =', img.shape)
print('Dimensão da imagem com o eixo de exemplos a mais =',
    imagens.shape)
```

```
Dimensão imagem original = (136, 205, 3)
Dimensão da imagem com o eixo de exemplos a mais = (1, 136,
 205, 3)
```

A imagem com eixo adicional está no tensor `imagens`.

Como fazer no TensorFlow-Keras

- O próximo passo é criar o gerador, instanciá-lo com a imagem e chamar 4 vezes para gerar as 4 novas imagens.

```
# Importa bibliotecas e funções necessárias
from tensorflow.keras.preprocessing.image import
    ImageDataGenerator
from matplotlib.pyplot import imshow, subplots, show

# Cria gerador
data_generator=ImageDataGenerator(rotation_range=60)

# Instancia gerador com o tensor de imagenes de entrada
image_iterator = data_generator.flow(imagens)

# Chama o gerador 4 vezes e mostra as 4 imagens geradas
fig, rows = subplots(nrows=1, ncols=4, figsize=(18,18))
for row in rows:
    row.imshow(image_iterator.next()[0].astype('int'))
    row.axis('off')
show()
```

Como fazer no TensorFlow-Keras

- As operações realizadas nesse código são as seguintes:
  - Primeiramente é importada a classe `ImageDatagenerator` do Keras
  - Também são importadas algumas funções da biblioteca matplotlib para mostrar as imagens
  - Nesse caso o gerador é criado com nome `data_generator` e, somente como exemplo, realiza transformação de rotação
  - O gerador é instanciado com a imagem desejada e associado á variável `image_iterator`
  - Novas imagens são geradas chamando o gerador com o método `next` (`image_iterator.next()`).
  - No loop `for` o gerador é chamado 4 vezes.

Como fazer no TensorFlow-Keras

- Para gerar imagens com transformações diferentes da rotação basta usar outros parâmetros ao criar o gerador.
- A única linha que deve ser alterada no código anterior para alterar o tipo de transformação é o comando para criar o gerador, ou seja:

```
# Cria gerador
data_generator = ImageDataGenerator(
    rotation_range=60)
```

Como fazer no TensorFlow-Keras

- **Mudança de escala:**

```
data_generator = ImageDataGenerator(zoom_range=0.4)
```

Uma mudança de tamanho aleatória é obtida pelo argumento `zoom_range` ⇒ nesse exemplo `zoom_range=0.4` significa redução/ampliação entre 0,6 e 1,4




Como fazer no TensorFlow-Keras

- **Inversão horizontal:**

```
data_generator = ImageDataGenerator(horizontal_flip=True)
```

Gera imagens invertidas horizontalmente de forma aleatoriamente

Como fazer no TensorFlow-Keras

- **Inversão vertical:**

```
data_generator = ImageDataGenerator(vertical_flip=True)
```

Gera imagens invertidas verticalmente de forma aleatoriamente

Como fazer no TensorFlow-Keras

- **Alteração de brilho:**

```
data_generator = ImageDataGenerator(
    brightness_range=(0.1,0.9))
```

`brightness_range` define o intervalo para escolher aleatoriamente um valor de mudança de brilho ⇒ 0 corresponde a uma imagem totalmente escura e 1 corresponde ao brilho igual à imagem original.




Como fazer no TensorFlow-Keras

- **Rotação:**

```
data_generator = ImageDataGenerator(rotation_range=90)
```

Deve-se definir intervalo de rotação (`rotation_range`) ⇒ imagem é girada aleatoriamente por um ângulo no intervalo de ±`rotation_range` em graus

Como fazer no TensorFlow-Keras

- **Translação horizontal:**

```
data_generator=ImageDataGenerator(width_shift_range=0.3)
```

`width_shift_range` é um número real entre 0 e 1 que define limite superior da fração da largura total pela qual a imagem é deslocada aleatoriamente, para a esquerda ou direita.




Como fazer no TensorFlow-Keras

- **Translação vertical:**

```
data_generator=ImageDataGenerator(height_shift_range=0.3)
```

Exatamente como na translação horizontal, exceto que a imagem é deslocada na vertical para cima ou para baixo

Como fazer no TensorFlow-Keras

- **Cisalhamento:**

```
data_generator=ImageDataGenerator(shear_range=45)
```

Transformação de cisalhamento inclina a imagem alongando-a em uma direção

`shear_range` especifica o ângulo da inclinação em graus que pode ser tanto positivo quanto negativo




Como fazer no TensorFlow-Keras

- **Combinação de várias transformações:**

```
data_generator=ImageDataGenerator(rotation_range=45,
    zoom_range=0.4,
    width_shift_range=0.2,
    height_shift_range=0.2,
    horizontal_flip=True)
```

Nesse caso as transformações são aplicadas simultaneamente de forma aleatória.




Técnicas de “data augmentation”

- **Preenchimento de lacunas:**

  Após as operações de transformações podem aparecer lacunas na imagem $\Rightarrow$ como preencher essas lacunas?

Técnicas de “data augmentation”

- **Preenchimento de lacunas:**
  1. Constante ⇒ usa um valor constante
  2. Borda ⇒ valores das bordas da imagem são estendidos nas lacunas
  3. Reflexão ⇒ valores de pixel da imagem são refletidos em torno da borda
  4. “Wrap” ⇒ porção da imagem perdida é repetida nas lacunas.

Como fazer no TensorFlow-Keras

- **Preenchimento de lacunas ⇒ borda:**

```
data_generator=ImageDataGenerator(width_shift_range=0.3,
                                  fill_mode='nearest')
```

Essa é a opção padrão ⇒ valor da borda mais próxima do pixel é repetido para todos os pixels da lacuna

Como fazer no TensorFlow-Keras

- **Preenchimento de lacunas ⇒ constante:**

```
data_generator=ImageDataGenerator(width_shift_range=0.3,
                fill_mode='constant', cval=190)
```

Preenche todos os pixels da lacuna por um valor constante, definido no parâmetro `cval`.

Como fazer no TensorFlow-Keras

- **Preenchimento de lacunas ⇒ reflexão:**

```
data_generator=ImageDataGenerator(width_shift_range=0.3,
                                  fill_mode='reflect')
```

Este modo cria uma "reflexão" e preenche as lacunas em uma ordem inversa dos valores conhecidos.




Como fazer no TensorFlow-Keras

- **Preenchimento de lacunas ⇒ “wrap”:**

```
data_generator=ImageDataGenerator(width_shift_range=0.3,
                                  fill_mode='wrap')
```

Cria efeito de 'quebra' copiando os valores dos pontos conhecidos, perdidos na transformação, nos pontos de lacuna, mantendo a ordem inalterada.




Uso de geração artificial de dados

- Observe que a classe `ImageDataGenerator` não realiza uma operação aditiva, ou seja, ela recebe as imagens originais, as transforma aleatoriamente e retorna somente as imagens transformadas e não as imagens originais junto com as transformadas
- Existem basicamente duas formas de usar geração artificial de dados para treinar uma RNA no contexto de processamento de imagens:
  - Expansão de um conjunto de imagens de forma a criar um novo conjunto de dados composto pelas imagens originais e as novas imagens criadas;
  - Geração de dados em tempo real durante o treinamento.

Expansão do conjunto de imagens

- Nesse caso se usa “data augmentation” para criar um novo conjunto de dados que é depois utilizado no treinamento de uma RNA.
- A partir de um conjunto de imagens geram-se novas imagens artificialmente por meio de transformações, criando um novo conjunto de dados maior, composto pelas imagens originais e as geradas artificialmente.
- Como fazer?

  Escolher um conjunto de transformações e usar o `ImageDataGenerator` para transformar as imagens de forma aleatória com o método `flow`.

Expansão do conjunto de imagens

- Similar ao exemplo de criar imagens artificiais a partir de uma imagem, repetindo esse processo para todas as imagens do conjunto de dados.
- As imagens criadas são salvas em arquivos para serem usadas posteriormente durante o treinamento da RNA.
- O novo conjunto de imagens é maior mas é fixo, ou seja, as mesmas imagens são usadas no treinamento em todas as épocas.
- Forma pouco utilizada:
  - O novo conjunto de imagens ocupa mais espaço em memória;
  - Mais trabalho e tempo para gerar esse novo conjunto de dados, principalmente se quiser uma quantidade grande de imagens.

Expansão do conjunto de imagens

- Primeiramente deve-se criar a lista de imagens do conjunto de dados originais que serão transformadas para criar as novas imagens.
- Essa operação é feita com funções das bibliotecas `os` e `globs`, como visto na aula sobre gerador de dados.

```
# Importa bibliotecas os e glob
from glob import glob
import os

# Define diretório das imagens desejadas
cat_path = 'cats_dogs/cats'

# Define tipo de arquivo a serem listados (jpeg)
glob_imgs_cat = os.path.join(cat_path, '*.jpg')

# Cria lista com os nomes dos arquivos
cat_img_paths = glob(glob_imgs_cat)
```

Expansão do conjunto de imagens

- O gerador de imagens artificial é criado com a classe `ImageDatagenerator` do Keras definindo as transformações desejadas.

```
# Importa classe ImageDataGenerator e numpy
from tensorflow.keras.preprocessing.image import
    ImageDataGenerator
import numpy as np

# Cria gerador artificial de imagens
datagen = ImageDataGenerator(
    rotation_range=30,
    width_shift_range=0.2,
    height_shift_range=0.2,
    horizontal_flip=True,
    fill_mode="nearest")
```

Nesse exemplo são escolhidas transformações de rotação, translação vertical e horizontal, e inversão horizontal.

Expansão do conjunto de imagens

- O passo final é criar um loop para carregar as imagens, instanciar o gerador e executá-lo quantas vezes for desejado para cada imagem.

```
# Número de novas imagens criadas para cada imagem original
n_img_transf = 2

# Loop para carregar as imagens originais e criar novas
imagens
for fname in cat_img_paths:

    # Carrega imagem do arquivo indicado no caminho fname
    img = imread(fname)

    # Adiciona eixo dos exemplos na imagem
    img = img.reshape((1, img.shape[0], img.shape[1],
        img.shape[2]))

    # Transforma imagem em um tensor numpy
    img = np.array(img)
```

Expansão do conjunto de imagens

```
(continuação)
# Instancia gerador com a imagem carregada
    aug_generator = datagen.flow(img,
        save_to_dir='cats_dogs/cats', save_prefix='cat',
        save_format='jpg')

    # Cria n_img_transf novas imagens de cada imagem do
    banco de dados
    for i in range(n_img_transf):
        aug_generator.next()
```

- `cat_img_paths` é a lista dos nomes dos arquivos com as imagens originais
- `save_to_dir` define diretório onde as imagens criadas serão salvas
- `save_prefix` define nome básico para os arquivos das imagens
- `save_format` define formato do arquivo da imagem (no caso é jpg)

Treinamento com geração artificial de dados

- Nesse caso escolhe-se um conjunto de transformações e usa o `ImageDataGenerator` para transformar as imagens de forma aleatória durante o treinamento da RNA juntamente com os métodos `flow_from_diretcory` e `fit_generator`.
- As imagens são criadas baseadas no conjunto de dados de treinamento, lote por lote, em tempo real durante o treinamento e logo após o seu uso são descartadas.
- As imagens criadas não são e nem precisam ser armazenadas em memória.
- Ressalta-se que o gerador de dados retorna somente as novas imagens criadas e não as originais junto com as transformadas.

Treinamento com geração artificial de dados

- Como as imagens são criadas aplicando transformações aleatórias, a cada época usam-se imagens diferentes $\Rightarrow$ isso aumenta a capacidade de generalização da RNA pois, a probabilidade de uma mesma imagem ser utilizada duas vezes durante o treinamento é quase nula.
- Uma pergunta que pode surgir $\Rightarrow$ Porque as imagens originais não são usadas? Elas não são úteis para o treinamento?
  - Um aspecto importante da geração artificial de dados é também garantir que a RNA receba somente imagens nunca vistas, com o objetivo de evitar problemas e “overfitting” e, assim, melhorar a sua capacidade de generalização;

Treinamento com geração artificial de dados

  - Se as imagens originais forem incluídas junto com as transformadas, em cada época a RNA receberia as imagens originais múltiplas vezes, reduzindo a sua capacidade de generalização.
  - Ressalta-se que um dos principais objetivos da geração artificial de dados é aumentar a capacidade de generalização da RNA.

- Na prática esse método faz com que a RNA apresente em geral um desempenho ligeiramente melhor para os dados de validação e teste do que para os dados de treinamento.
- Essa é a forma mais utilizada e mais eficiente $\Rightarrow$ pois exige menos trabalho que o método de expansão do conjunto de dados (único trabalho é definir o gerador de dados) e aumenta a capacidade de generalização da RNA.

Treinamento com geração artificial de dados

- A primeira etapa desse processo é criar os gerados de dados de treinamento e de validação/teste.

```
# Importa classe ImageDatagenerator
from tensorflow.keras.preprocessing.image import
ImageDataGenerator

# Cria gerador de dados de treinamento com "data
augmentation"
train_datagen = ImageDataGenerator(rescale=1./255.,
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    horizontal_flip=True,
    fill_mode="nearest")

# Cria gerador de dados de validação e teste sem "data
augmentation"
val_test_datagen = ImageDataGenerator(rescale=1./255.)
```

Treinamento com geração artificial de dados

- Nesse caso somente o gerador de dados de treinamento deve realizar “data augmentation”.
- Os geradores de dados de validação e teste devem usar sempre as imagens originais normalizadas.
- Para normalizar as imagens, ambos os geradores devem transformar os valores dos pixels de números inteiros entre 0 e 255 para números reais entre 0 e 1, dividindo por 255 $\Rightarrow$ isso é definido pelo parâmetro `rescale=1./255.`
- Nesse exemplo, o gerador de dados de treinamento utiliza as transformações de rotação, translação vertical, translação horizontal e inversão horizontal.

Treinamento com geração artificial de dados

- O próximo passo é instanciar os geradores de treinamento, validação e teste.
- Nessa operação temos que passar os respetivos diretórios onde estão as imagens e definir os parâmetros: tamanho do lote (`batch_size`), dimensão das imagens (`target_size`) e tipo de problema (`class_mode`).

```
# Define dimensão das imagens
img_size = (150, 150)

# Instancia o gerador das imagens treinamento
train_generator = train_datagen.flow_from_directory(
    'cats_and_dogs_small/train/',
    target_size=img_size,
    batch_size=20,
    class_mode='binary')
```

Treinamento com geração artificial de dados

```
# Inicializa o gerador com as imagens de validação
val_generator = val_test_datagen.flow_from_directory(
    'cats_and_dogs_small/val',
    target_size=img_size,
    batch_size=20,
    class_mode='binary')

# Inicializa o gerador com as imagens de validação
test_generator = val_test_datagen.flow_from_directory(
    'cats_and_dogs_small/test',
    target_size=img_size,
    batch_size=20,
    class_mode='binary')
```

- Nesse exemplo tem-se:
  - Dimensão das imagens: `target_size = (150, 150)`
  - Tamanho do lote: `batch_size = 20`
  - Tipo de problema: `class_mode = binary`

Treinamento com geração artificial de dados

- O treinamento da RNA é realizado usando o método `fit_generator` usando os geradores de dados de treinamento e validação.

```
history = rna.fit_generator(train_generator,
                    validation_data=val_generator,
                    steps_per_epoch=40,
                    epochs=50,
                    validation_steps=10,
                    verbose=2)
```

  - Nesse exemplo são usadas 50 épocas de treinamento.
  - 40 passos são necessários para carregar todas as imagens de treinamento transformadas e 10 passos para carregar as imagens de validação.

Treinamento com geração artificial de dados

- Para avaliar o desempenho da RNA treinada é usado o método `evaluate_generator` com o gerador de dados de teste.

```
# Calcula a função de custo e a métrica para os dados de teste
custo_metrica_test =
    rna.evaluate_generator(test_generator, steps=10)
```

  - 10 passos são necessários para carregar todos as imagens de teste.

- **Observação:** no caso de “data augmentation” o número de `steps_per_epoch` usado no treinamento pode ser maior do que o número necessário para carregar todas as imagens de treinamento, nesse caso o gerador vai continuar a criar novas imagens.