Zajęcie 8. Uczenie głębokie w R. Klasyfikator obrazów za pomocą Keras

Abstract

Celem jest uczenie głębokie za pomocą R i pakietu keras

1. Wstęp

Oto strona projektu Keras `https://keras.io/`
Dokładna specyfikacja biblioteki `https://keras.io/api/`
Opis zbiorów danych Keras [2]
Przykład użycia w R `https://appsilon.com/r-keras-mnist/`
Przykład użycia w Python - kod
Oto książka w języku polskim [1].

2. Zainstalowanie Tensorflow i Keras

2.1. R:

TensorFlow:

```
install.packages("tensorflow")
library(tensorflow)
install_tensorflow()
```

keras:

```
install.packages("keras")
library(keras)
install_keras()
```

2.2. Python

```
import numpy as np
import keras
from keras import layers
```

Tensorflow musi być zainstalowny pierwszym!

3. Ładowanie i przygotowanie zestawu danych

Zbiór danych MNIST jest wbudowany w bibliotekę Keras. Możesz to uzyskać, wywołując funkcję `dataset_mnist()` po zaimportowaniu biblioteki.

Ponadto należy podzielić zbiór danych na cztery kategorie:

- X_train - zawiera cyfry ze zbioru uczącego
- X_test - zawiera cyfry dla zestawu testowego
- y_train - zawiera etykiety zestawu uczącego
- y_test - zawiera etykiety zestawu testowego

Poniższy fragment kodu yżywamy, aby zaimportować Keras i rozpakować dane:

3.0.1. R:

```
library(keras)
mnist <- dataset_mnist()
X_train <- mnist$train$x
X_test <- mnist$test$x
y_train <- mnist$train$y
y_test <- mnist$test$y
```

3.0.2. Python:

```
 # Model / data parameters
num_classes = 10
input_shape = (28, 28, 1)

# Load the data and split it between train and test sets
(x_train, y_train), (x_test, y_test) = keras.datasets.mnist.load_data()
```

3.1. Obraz wejściowy do warstwy liniowej

Żeby zmienić kształt obrazów wejściowych np. z 28×28 na 1×784 każdy, możemy to zrobić za pomocą funkcji `array_reshape()` z Keras. Ponadto podzielimy również każdą wartość matrycy obrazu przez 255, więc wszystkie obrazy będą należeć do zakresu [0, 1].

To poradzi sobie z obrazami wejściowymi, ale musimy również przekonwertować etykiety. Są one domyślnie przechowywane jako liczby całkowite i przekonwertujemy je na kategorie za pomocą funkcji `to_categorical()`.

Oto kod

3.1.1. R:

```
X_train <- array_reshape(X_train, c(nrow(X_train), 784))
X_train <- X_train / 255

X_test <- array_reshape(X_test, c(nrow(X_test), 784))
X_test <- X_test / 255

y_train <- to_categorical(y_train, num_classes = 10)
y_test <- to_categorical(y_test, num_classes = 10)
```

3.1.2. Python:

```
# Scale images to the [0, 1] range
x_train = x_train.astype("float32") / 255
x_test = x_test.astype("float32") / 255
# Make sure images have shape (28, 28, 1)
x_train = np.expand_dims(x_train, -1)
x_test = np.expand_dims(x_test, -1)
print("x_train shape:", x_train.shape)
print(x_train.shape[0], "train samples")
print(x_test.shape[0], "test samples")
```

3.2. Obraz wejściowy do warstwy spłaszczenia

Podzielimy każdą wartość matrycy obrazu przez 255, więc wszystkie obrazy będą należeć do zakresu [0, 1].

Musimy również przekonwertować etykiety. Są one domyślnie przechowywane jako liczby całkowite i przekonwertujemy je na kategorie za pomocą funkcji `to_categorical()`.

Oto kod

3.2.1. R:

```
X_train <- X_train / 255

X_test <- X_test / 255

y_train <- to_categorical(y_train, num_classes = 10)
y_test <- to_categorical(y_test, num_classes = 10)
```

3.2.2. Python:

```
# convert class vectors to binary class matrices
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)
```

4. Trening modelu

4.1. Tworzenie architektury modelu w R

4.1.1. Warstwa liniowa

Będziemy mieć trzy ukryte warstwy z odpowiednio 256, 128 i 64 neuronami oraz warstwę wyjściową z dziesięcioma neuronami, ponieważ zbiór danych MNIST zawiera dziesięć różnych klas.

Po każdej warstwie liniowej następuje `layer_dropout`[1], aby zapobiec nadmiernemu dopasowaniu.

Oto kod:

```
model <- keras_model_sequential() %>%
  layer_dense(units = 256, activation = "relu", input_shape = c(784)) %>%
  layer_dropout(rate = 0.25) %>%
  layer_dense(units = 128, activation = "relu") %>%
  layer_dropout(rate = 0.25) %>%
  layer_dense(units = 64, activation = "relu") %>%
  layer_dropout(rate = 0.25) %>%
  layer_dense(units = 10, activation = "softmax")
```

[1] część wyjść (mianowicie `rate`) po warstwie dropout losowo zostanie zerowana, czyli usuwa się część neuronów

4.1.2. Warstwa spłaszczenia

```
model <- keras::keras_model_sequential() %>%
  layer_flatten(input_shape = c(28, 28)) %>%
  layer_dense(units = 128, activation = 'relu') %>%
  layer_dense(units = 10, activation = 'softmax')
```

4.1.3. Wyświetlenie architektury modelu

Po zadeklarowaniu modelu możemy użyć funkcji `summary ()`, aby wydrukować jego architekturę:

```
summary(model)
```

4.2. Tworzenie modelu w Python

```
model = keras.Sequential(
    [
        keras.Input(shape=input_shape),
        layers.Conv2D(32, kernel_size=(3, 3), activation="relu"),
        layers.MaxPooling2D(pool_size=(2, 2)),
        layers.Conv2D(64, kernel_size=(3, 3), activation="relu"),
        layers.MaxPooling2D(pool_size=(2, 2)),
        layers.Flatten(),
        layers.Dropout(0.5),
        layers.Dense(num_classes, activation="softmax"),
    ]
)
model.summary()
```

4.3. Kompilowanie modelu

Ten krok obejmuje wybór sposobu mierzenia straty, wybór funkcji zmniejszającej stratę oraz wybór miernika, który mierzy ogólną wydajność.

Oto przykład:

4.3.1. R:

```
model %>% compile(
  loss = "categorical_crossentropy",
  optimizer = optimizer_adam(),
  metrics = c("accuracy")
)
```

4.3.2. Python:

```
model.compile(loss="categorical_crossentropy", optimizer="adam",
                                                metrics=["accuracy"])
```

4.4. Trenowanie modelu

Możemy teraz wywołać funkcję `fit()`, aby wytrenować model. Poniższy fragment uczy model przez 50 epok, dostarczając jednocześnie 128 obrazów:

4.4.1. R:

```
history <- model %>%
  fit(X_train, y_train, epochs = 50, batch_size = 128, validation_split = 0.15)
```

4.4.2. Python:

```
batch_size = 128
epochs = 15
model.fit(x_train, y_train, batch_size=batch_size, epochs=epochs,
                                                validation_split=0.1)
```

5. Ocena modelu

Możemy użyć funkcji `evaluate()` z Keras, aby ocenić wydajność na zbiorze testowym. Oto fragment kodu, który to robi:

5.1. R:

```
model %>%
  evaluate(X_test, y_test)
```

5.2. Python:

```
score = model.evaluate(x_test, y_test, verbose=0)
print("Test loss:", score[0])
print("Test accuracy:", score[1])
```

6. Prognozowanie

Aby przewidzieć nowy podzbiór danych, możesz użyć funkcji `predict_classes()`, jak pokazano poniżej:

6.1. R:

```
model %>%
  predict_classes(X_test)
```

6.2. Python:

```
prediction = model.predict(x_test)

import matplotlib.pyplot as plt

img = plt.imshow(1-x_test[1])
img.set_cmap('gray')
plt.axis('off')
plt.show()
```

7. Warianty Zadania

Uwaga! Sprawozdania muszą być sporządzane zgodnie ze wzorem. Oprócz tego pliki źródłowe oraz obrazy muszą być zachowane w zdalnym repozytorium.

Zadanie dotyczy konstruowania sieci głębokiej w celu klasyfikacji obrazów pobranych ze zbioru danych. Warianty zadania są określone zbiorem danych obrazów, który może być pobrany na stronie `https://keras.io/api/datasets/`

1. CIFAR-10
2. CIFAR-100
3. MNIST database of handwritten digits
4. Fashion-MNIST database of fashion articles
5. CIFAR-10
6. CIFAR-100

7. MNIST database of handwritten digits

8. Fashion-MNIST database of fashion articles

9. CIFAR-10

10. CIFAR-100

11. MNIST database of handwritten digits

12. Fashion-MNIST database of fashion articles

References

[1] , ???? Deep learning. praca z językiem r i biblioteką keras. książka, ebook. francois chollet, j. j. allaire. księgarnia informatyczna helion.pl. https://helion.pl/ksiazki/deep-learning-praca-z-jezykiem-r-i-biblioteka-keras-francois-chollet-j-j-allaire-delerk.htm#format/d, (Accessed on 05/07/2021).

[2] , ???? Exploring the keras datasets – machinecurve. https://www.machinecurve.com/index.php/2019/12/31/exploring-the-keras-datasets/, (Accessed on 05/07/2021).