Metody sztucznej inteligencji
Keras

Prowadzący:
Jakub Szyguła: jakub.szygula@polsl.pl
Dariusz Marek: dariusz.marek@polsl.pl

30 listopada 2019

Spis treści

1 Wprowadzenie

Keras to biblioteka sieci neuronowych. Może działać z TensorFlow, Microsoft Cognitive Toolkit, Theano lub PlaidML. Umożliwia szybkie eksperymentowanie z głębokimi sieciami neuronowymi.

1.1 Wymagania

Do działania biblioteki Keraz wraz z TensorFlow wymagane jest:

1. Python 3.7.4 https://www.python.org/downloads/release/python-374/
2. TensorFlow v.2.0
3. Keras
4. Plaidml - * Opcjonalnie, umożliwia wybór procesora/karty graficznej do obsługi tensorflow
5. Dodanego Python'a do Path w zmiennych środowiskowych systemu Windows np.:
   (a) $C:\backslash Python37\backslash$
   (b) $C:\backslash Python37\backslash Scripts\backslash$

1.2 Instalacja modułów

Instalacja biblioteki Keras za pomocą modułu pip do Python'a. W konsoli (cmd) należy wpisać:

```
pip install keras
```

Instalacja biblioteki TensorFlow

```
pip install tensorflow==2.0
```

1.3 Plaidml

Instalacja biblioteki Plaidml

```
pip install plaidml-keras plaidbench
```

Konfiguracja biblioteki Plaidml

```
plaidml-setup
```

Po uruchomieniu komendy, uruchamia się konsolowa aplikacja umożliwiająca wybór układu używanego do obliczeń. Następnie należy włączyć eksperymentalne wspieranie układów [y]. Kolejno wybrać właściwe urządzenie i zapisać.

Następnie w pliku gdzie uruchamiany jest tensorflow i keras należy dadać dodatkową linię zmieniającą ustawienia tensorflowa. Linię należy dodać przed pierwszym oddwołaniem się do keras'a i tensorflow'a.

```
import os
os.environ["KERAS_BACKEND"] = "plaidml.keras.backend"
```

2 Wprowadzenie do Keras'a

2.1 Warstwy

1. Warstwy gęste (Dense) są do danych wektorowych (dwuwymiarowe tensory).
2. Warstwy LSTM - do danych sekwencyjnych (świetne do zastosowania dla pasów transmisyjnych, (trójwymiarowe tensory).
3. Warstwy Konwolucyjne - dla danych obrazu (tensory czterowymiarowe).

3 Modele

1. Sequential (liniowe stosy warstw, obecnie najczęściej spotykany) – pozwala na tworzenie modeli warstwa po warstwie.
2. Funkcjonalny interfejs API (acykliczne grafy warstw) - pozwala na tworzenie modeli, które łączą się nie tylko z poprzednią i następną warstwą, a także mogą mieć wiele wejść lub wyjść.

4 Funkcje aktywacji

Funkcja uzywana do obliczannia wartość wyjścia z neuronów.

1. Sigmoid – normalizuje wartości do zakresu 0 do 1, co pozwala interpretować otrzymane wyniki jako prawdopodobieństwo.
2. TanH – w stosunku do Sigmoida pozwala na unikanie stronniczości gradientu przez szerszy zakres wartości: -1 do 1. (TanH była najpopularniejszą funkcją aktywacji na początku rozwoju sieci neuronowych).
3. Softmax - jest to tak naprawdę funkcja wykładnicza. Jej wartość zostaje znormalizowana w taki sposób, aby suma aktywacji dla całej warstwy wynosiła 1. Stosowana najczęściej w warstwie wyjściowej do rozwiązywania problemów klasyfikacyjnych.
4. Relu – ma za zadanie „wyzerować" wartości negatywne (obecnie najczęściej stosowana funkcja aktywacji). Relu zwana jest również funkcją nieliniową.

5 Funkcje celu

Funkcje celu należy dobierać zależnie od problemu, w bardzo specyficznych wypadkach można tworzyć takie funkcje samodzielnie.

Rodzaj problemu	Funkcja aktywacji ostatniej warstwy	Funckcja straty
Klasyfikacja binarna	sigmoid	binary_crossentropy
Wieloklasowa klasyfikacja jednoetykietowa	softmax	categorical_crossentropy
Wieloklasowa klasyfikacja wieloetykietowa	sigmoid	binary_crossentropy
Regresja dowolnych wartości	Brak	mse
Regresja wartości z zakresu od 0 do 1	sigmoid	mse lub binary_crossentropy

5.1 Dokumentacja

1. Keras https://keras.io/
2. Funkcje straty (celu): https://keras.io/losses/
3. Funkcje aktywacji: https://keras.io/activations/
4. Optymalizatory: https://keras.io/optimizers/

6 Przykład 1 - Sieć rozpoznająca cyfry

Ładowanie potrzebych modułów MNIST - zbór obazów z odręcznie pisanymi cyframi od 0 do 9 Sequential- model sekwencyjny sieci neuronowej Dense - warsta gęsta sieci

```
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense
from keras.utils import np_utils
```

Wczytywanie danych

```
(X_train, y_train), (X_test, y_test) = mnist.load_data()
```

Splaszczenie obrazów z28*28 pikseli do 784 elementowego vector'a

```
num_pixels = X_train.shape[1] * X_train.shape[2]
X_train = X_train.reshape((X_train.shape[0], num_pixels)).
 ↪astype('float32')
X_test = X_test.reshape((X_test.shape[0], num_pixels)).astype('float32')
```

Normalizacja danych o wartosciach od 0 do 255 do wartości od 0 do 1

```
X_train = X_train / 255
X_test = X_test / 255
```

Pobranie i stworzenie listy klas dla danych

```
y_train = np_utils.to_categorical(y_train)
y_test = np_utils.to_categorical(y_test)
```

Wyciągniecie liczby klas

```
num_classes = y_test.shape[1]
```

Tworzenie modelu sieci

```
model = Sequential()
```

Dodanie pierwszej warsty odpowiedzialnej za odebranie danych obrazu - liczba neuronow = liczbie pikseli

```
model.add(Dense(num_pixels, input_dim=num_pixels,␣
↪kernel_initializer='normal', activation='relu'))
```

Dodanie drugiej warstwy odpowiedzialnej za klasę - liczbla neuronow = liczba klas

```
model.add(Dense(num_classes, kernel_initializer='normal',␣
↪activation='softmax'))
```

Kompilacja modelu

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

Uczenie modelu danymi

```
model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=10,␣
↪batch_size=200, verbose=2)
```

Testowanie modelu

```
scores = model.evaluate(X_test, y_test, verbose=0)
print("Baseline Error: %.2f%%" % (100-scores[1]*100))
```

7 Przykład 2 - Sieć z warstwami konwolucyjnymi rozpoznająca cyfry

Ładowanie potrzebnych modułów MNIST - zbiór obrazów z odręcznie pisanymi cyframi od 0 do 9 Sequential- model sekwencyjny sieci neuronowej

```
from keras.datasets import mnist
from keras.utils import np_utils
from keras import layers
from keras import models
from keras.utils import to_categorical
```

Wczytywanie danych

```
(X_train, y_train), (X_test, y_test) = mnist.load_data()
```

Przekształcanie wielkości obrazów do 28x28x1 pixel oraz ich normalizacja

```
train_images = train_images.reshape((60000, 28, 28, 1))
train_images = train_images.astype('float32') / 255

test_images = test_images.reshape((10000, 28, 28, 1))
test_images = test_images.astype('float32') / 255
```

Pobranie i stworzenie listy klas dla danych

```
train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)
```

tworzenie modelu sieci

```
model = models.Sequential()
```

Dodanie pierwszej warsty konwolucyjnej złożonej z 32 kerneli o wielkości 3x3

```
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28,␣
↪28, 1)))
```

Dodanie warstwy zmiejszającej wielkość powstałych obrazów z warstwy konwolucyjnej

```
model.add(layers.MaxPooling2D((2, 2)))
```

Dodanie drugiej warstwy konwolucyjnej

```
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
```

Dodanie warstwy spłaszczającej dane 2D do 1D

```
model.add(layers.Flatten())
```

Dodanie warstwy gęstej odpowiedzialnej za klasę - liczbla neuronow = liczba klas

```
model.add(layers.Dense(10, activation='softmax'))
```

Kompilacja modelu

```
model.compile(optimizer='rmsprop',
loss='categorical_crossentropy',
metrics=['accuracy'])
```

Włączenie uczenia sieci

```
model.fit(train_images, train_labels, epochs=5, batch_size=64)
```

Testowanie modelu

```
test_loss, test_acc = model.evaluate(test_images, test_labels)
print(test_loss, test_acc)
```

8 Zadania do wykonania

8.1 Zadanie 1

1. Zapoznaj się z funkcjonalnością biblioteki Keras, sprawdz jej działanie, dodaj kolejne warstwy sieci i sprawdz rezultaty.

2. Korzystając z Internetu i dokumentacji dodaj kod odpowiedzialny za zapis/odczyt nauczonej sieci do/z pliku.

8.2 Zadanie 2

Napisz program, który załaduje narysowany np. w Paint obraz i dokona jego klasyfikacji.

Po wczytaniu obrazu wykorzystaj funkcję:
def rgb2gray(rgb):
    return np.dot(rgb[...,:3], [0.2989, 0.5870, 0.1140])

8.3 Zadanie 3

Korzystając z Internetu i dokumentacji wykorzystaj drugi przykładowy zbiór mikro zdjęć wbudowany w Keras'a: CIFAR-10 i stwórz model do rozpoznawania dostępnych w nim obiektów.
CIFAR-10 to baza 60 000 kolorowych obrazów o rozmiarze 32×32 piksele wraz z etykietami przypisującymi je do 10 rozłącznych klas (airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck).