Baza danych MNIST

Przedmiot: **Uczenie Maszynowe**

Prowadzący: **dr inż. Pietroń Marcin**

dr inż. Wielgosz Maciej

Kierunek: **Informatyka i Ekonometria**

Wykonanie: **Kamil Polaczek**

Kraków 2019

Baza danych MNIST (Modified National Institute of Standards and Technology database) ręcznie napisanych cyfr składa się z zestawu szkoleniowego 60 000 przykładów i zestawu testowego obejmującego 10 000 przykładów. Jest to podzbiór większego zestawu dostępnego w NIST. Ponadto czarno-białe obrazy z NIST zostały znormalizowane i wyśrodkowane, aby zmieścić się w 28x28 pikseli i wygładzonej.

Ta baza danych jest bardzo popularna w szkoleniach i testach w dziedzinie uczenia maszynowego i przetwarzania obrazu. Jest to zremiksowany podzbiór oryginalnych zbiorów danych NIST. Połowa z 60 000 obrazów treningowych składa się z obrazów z zestawu danych testowych NIST, a druga połowa z zestawu treningowego Nist. 10 000 obrazów z zestawu testowego jest podobnie zmontowanych.

Zestaw danych MNIST jest wykorzystywany przez naukowców do testowania i porównywania wyników badań z innymi. Najniższe wskaźniki błędów w literaturze wynoszą zaledwie 0,21 procent.

Niektórzy badacze osiągnęli „niemal ludzką wydajność" w bazie danych MNIST. Najwyższy wymieniony wskaźnik błędu na oryginalnej stronie internetowej bazy danych wynosi 12 procent, co osiąga się przy użyciu prostego klasyfikatora liniowego bez przetwarzania wstępnego.

W 2004 r. Naukowcy wykorzystali nowy wskaźnik błędu o wartości 0,42 procent w bazie danych, wykorzystując nowy klasyfikator o nazwie LIRA, który jest klasyfikatorem neuronowym z trzema warstwami neuronów opartymi na zasadach perceptronu Rosenblatta.

Niektórzy badacze przetestowali systemy sztucznej inteligencji z wykorzystaniem bazy danych poddanej przypadkowym zniekształceniom. Systemy w tych przypadkach są zwykle sieciami neuronowymi, a stosowane zniekształcenia są albo zniekształceniami afinicznymi, albo zniekształceniami elastycznymi. Czasami systemy te mogą być bardzo skuteczne; jeden taki system osiągnął wskaźnik błędu w bazie danych wynoszący 0,39 proc.

W 2011 r. Naukowcy korzystali z podobnego systemu sieci neuronowych, uzyskując wskaźnik błędu wynoszący 0,27%, poprawiając poprzedni najlepszy wynik. W 2013 r. Twierdzono, że podejście oparte na uregulowaniu sieci neuronowych za pomocą DropConnect osiąga współczynnik błędu 0,21%. Niedawno najlepsza wydajność pojedynczej splotowej sieci neuronowej wynosiła 0,31 procent błędu. Od sierpnia 2018 r. najlepsza wydajność pojedynczej splotowej sieci neuronowej przeszkolonej w danych treningowych MNIST przy użyciu rozszerzania danych w czasie rzeczywistym wynosi 0,26% wskaźnika błędu. Ponadto Parallel Computing Center (Khmelnitskiy, Ukraina) uzyskało zespół tylko 5 splotowych sieci neuronowych, które działają na MNIST przy 0,21 procentowym poziomie błędu.

Projekt rozpoczyna się od zaimportowania potrzebnych klas i funkcji.

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

Następnie zainicjowano generator liczb losowych do stałej, aby zapewnić powtarzalność wyników skryptu.

```
seed = 7
numpy.random.seed(seed)
```

Załadowany został zestaw danych MNIST za pomocą funkcji pomocniczej Keras.

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

Zestaw danych szkoleniowych ma strukturę trójwymiarową instancji, szerokości obrazu i wysokości obrazu. W przypadku wielowarstwowego modelu perceptronu musimy zredukować obrazy do wektora pikseli. W tym przypadku obrazy o rozmiarze 28 × 28 będą miały wartości wejściowe 784 piksele.

Wykonano zmiane za pomocą funkcji reshape () w tablicy NumPy.

```
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')
```

Wartości pikseli to skala szarości od 0 do 255.

```
# znormalizuje wejścia od 0-255 do 0-1
X_train = X_train / 255
X_test = X_test / 255
```

Ostatecznie zmienna wyjściowa jest liczbą całkowitą od 0 do 9 więc przekształcony został wektor liczb całkowitych w macierz binarną.

Wykonane to zostało za pomocą wbudowanej funkcji pomocniczej np_utils.to_categorical () w Keras.

```
y_train = np_utils.to_categorical(y_train)
y_test = np_utils.to_categorical(y_test)
num_classes = y_test.shape[1]
```

Tworzymy prosty model sieci neuronowej.

```
def baseline_model():

        model = Sequential()
        model.add(Dense(num_pixels, input_dim=num_pixels, kernel_initializer='normal',
activation='relu'))
        model.add(Dense(num_classes, kernel_initializer='normal', activation='softmax'))

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

Model jest prostą siecią neuronową z jedną ukrytą warstwą o takiej samej liczbie neuronów, ile jest wejść (784). Funkcja aktywacji prostownika jest używana dla neuronów w ukrytej warstwie.

Funkcja aktywacji softmax jest używana na warstwie wyjściowej, aby przekształcić dane wyjściowe w wartości podobne do prawdopodobieństwa i pozwolić na wybranie jednej klasy z 10 jako przewidywania wyników modelu. Utrata logarytmiczna jest wykorzystywana jako funkcja straty (called categorical_crossentropy in Keras), a wydajny algorytm gradientu ADAM jest używany do uczenia wag.

Możemy teraz dopasować i ocenić model. Model pasuje do ponad 10 epok z aktualizacjami co 200 obrazów. Dane testowe są używane jako zestaw danych walidacyjnych, co pozwala zobaczyć umiejętności modelu podczas treningu. Szczegółowa wartość 2 jest używana do zmniejszenia wyniku do jednego wiersza dla każdej epoki treningowej.

Na koniec zestaw danych testowych jest używany do oceny modelu i drukowany jest wskaźnik błędu klasyfikacji.

```
model = baseline_model()

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

scores = model.evaluate(X_test, y_test, verbose=0)
```

print("Blad: %.2f%%" % (100-scores[1]*100))

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/10
 - 6s - loss: 0.2811 - acc: 0.9207 - val_loss: 0.1415 - val_acc: 0.9569
Epoch 2/10
 - 7s - loss: 0.1116 - acc: 0.9677 - val_loss: 0.0916 - val_acc: 0.9717
Epoch 3/10
 - 7s - loss: 0.0715 - acc: 0.9799 - val_loss: 0.0784 - val_acc: 0.9773
Epoch 4/10
 - 6s - loss: 0.0505 - acc: 0.9857 - val_loss: 0.0748 - val_acc: 0.9764
Epoch 5/10
 - 7s - loss: 0.0372 - acc: 0.9892 - val_loss: 0.0684 - val_acc: 0.9785
Epoch 6/10
 - 6s - loss: 0.0270 - acc: 0.9927 - val_loss: 0.0623 - val_acc: 0.9804
Epoch 7/10
 - 6s - loss: 0.0207 - acc: 0.9947 - val_loss: 0.0614 - val_acc: 0.9812
Epoch 8/10
 - 7s - loss: 0.0140 - acc: 0.9970 - val_loss: 0.0609 - val_acc: 0.9805
Epoch 9/10
 - 6s - loss: 0.0107 - acc: 0.9977 - val_loss: 0.0578 - val_acc: 0.9816
Epoch 10/10
 - 7s - loss: 0.0081 - acc: 0.9985 - val_loss: 0.0610 - val_acc: 0.9811
Blad: 1.89%
```

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


seed = 7
numpy.random.seed(seed)

# ładuj dane
(X_train, y_train), (X_test, y_test) = mnist.load_data()


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')

# znormalizuje wejścia od 0-255 do 0-1
X_train = X_train / 255
X_test = X_test / 255




y_train = np_utils.to_categorical(y_train)
y_test = np_utils.to_categorical(y_test)
num_classes = y_test.shape[1]

# model
def baseline_model():

        model = Sequential()
        model.add(Dense(num_pixels, input_dim=num_pixels, kernel_initializer='normal',
activation='relu'))
        model.add(Dense(num_classes, kernel_initializer='normal', activation='softmax'))

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


model = baseline_model()

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

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