Rafał Skrzypiec - Keras

Modele sieci neuronowych opisywane w tej pracy zostały zaprogramowane przy użyciu biblioteki Keras. Keras jest interfejsem API wysokiego poziomu służącym do tworzenia i szkolenia modeli głębokiego uczenia. Początkowo Keras został opracowany dla naukowców, którzy mogli dzięki niemu dokonywać szybkich eksperymentów i symulacji. Dzięki temu, że jest rozpowszechniany pod licencją MIT, co oznacza, że może być za darmo wykorzystywany w projektach komercyjnych, zdobył dużą popularność. Dziś ma on kilka set tysięcy użytkowników, od nauczycieli akademickich po inżynierów oprogramowania pracujących zarówno w start-upach jak i dużych firmach, i hobbystów. Jego zalety są wykorzystywane między innymi w wiodących ośrodkach naukowych takich jak Europejska Organizacja Badań Jądrowych CERN i setkach firm, z których największe to Google, Netflix, Uber, Yelp, Opera Software. Kaggle to platforma internetowa, która organizuje konkursy na najlepsze modele służące do przewidywania i opisywania zbiorów danych przesyłanych przez firmy i użytkowników. Jednym z najpopularniejszych narzędzi wykorzystywanych przez analityków jest Keras, wiele z konkursów zostało wygranych przez modele zbudowane przy użyciu wspomnianego interfejsu API.

Do największych zalet Keras należą:

- posiada przyjazny użytkownikowi interfejs, który ułatwia szybkie prototypowanie modeli sieci neuronowych
- prosty i spójny interfejs zoptymalizowany pod kątem typowych przepadków użycia
- zapewnia przejrzyste informacje zwrotne dotyczące błędów użytkownika
- obsługuje dowolne architektury sieciowe: modele z wieloma wejściami lub wieloma wyściami
- posiada wbudowane wsparcie dla splotowych sieci neuronowych oraz rekurencyjnych sieci neuronowych
- pozwala na bezproblemowe działanie tego samego kodu na CPU oraz GPU

Keras jest biblioteką, o której można powiedzieć, że zapewnia cegły służące do zbudowania modelu głębokiego uczenia natomiast w minimalnym stopniu pozwala użytkownikom na ingerencję w ich strukturę. W zamian wykorzystuje wyspecjalizowaną i dobrze zoptymalizowaną bibliotekę wyspecjalizowaną w operacjach na tensorach. Szczególnie szybko wykonują się obliczenia numeryczne typowe dla algorytmów uczenia maszynowego takich jak mnożenie macierzy i obliczanie gradientu. Można wybierać wśród trzech istniejących implementacji, każda z nich ma otwarte źródło. Pierwsza z nich wykorzystuje Tensorflow opracowany i rozwijany przez Google'a, druga korzysta z Theano opracowanego i rozwijanamegp przez LISA Lab w Uniwersytecie Montrealskim, ostatnia i najmniej popularna wykorzystuje CNTK opracowane i rozwijane przez Microsoft.

W przyszłości prawdopodobnie pojawi się więcej możliwości wyboru, między innymi niedawno powstały, zdobywający coraz większą popularność projekt Torch finansowany przez Facebooka. Obecnie najczęściej wykorzystywany jest TensorFlow, został on także wykorzystany w tej pracy.

Poniżej zaprezentuję jak proste jest zbudowanie i wytrenowanie bardzo podstawowego przykładu sieci neuronowej przy użyciu biblioteki Keras. Cały proces wymaga wykonania kilku kroków:

1. Zdefiniuj swoje dane treningowe: dane wejściowe i dane wyjściowe
2. Zdefiniuj warstwy swojej sieci neuronowej, które przekształcają dane wyjściowe w wyjście
3. Skonfiguruj proces uczenia poprzez wybranie funkcji straty, algorytmu szukającego minimum funkcji straty
4. Przeprowadź odpowiednią do wytrenowania sieci ilość iteracji

Zdefiniowana poniżej sieć składa się z dwóch warstw ukrytych o odpowiednio 10 i 5 neuronach ukrytych. Funkcją aktywacji w pierwszej warstwie jest sigmoida, dane wejściowe zawierają dwie cechy, które posłużą do zbudowania modelu, druga warstwa wykorzystuje tangens hiperboliczny jako funkcję aktywacji. Model podczas nauki minimalizuje błąd średniokwadratowy, wykorzystuje do tego algorytm rmsprop, trenownanie modelu skończy się po 100 pełnych iteracjach zbioru danych.

```
#Zaimportuj wymagane pliki
from keras import models
from keras import layers

#Zainicjalizuj model
model = models.Sequential()

#Dodaj pierwszą warstwę
model.add(layers.Dense(units = 10, activation = 'sigmoid',
    input_shape = 2))

#Dodaj drugą warstwę
model.add(layers.Dense(units = 5, activation = 'tanh'))

#Dodaj warstwę wyjsciową
model.add(layers.Dense(units = 1))

#Skompiluj model
model.compile(optimizer = 'rmsprop', loss='mse')

#Trenuj model
model.fit(inputs = X, outputs = Y, epochs = 100)
```