Helion

TensorFlow

13 praktycznych projektów
wykorzystujących uczenie maszynowe

Packt›

Ankit Jain, Armando Fandango, Amita Kapoor

Tytuł oryginału: TensorFlow Machine Learning Projects: Build 13 real-world projects with advanced numerical computations using the Python ecosystem

Tłumaczenie: Leszek Sagalara

ISBN: 978-83-283-5708-2

Helion SA
ul. Kościuszki 1c, 44-100 Gliwice
tel. 32 231 22 19, 32 230 98 63
e-mail: *helion@helion.pl*
WWW: *http://helion.pl* (księgarnia internetowa, katalog książek)

Pliki z przykładami omawianymi w książce można znaleźć pod adresem:
ftp://ftp.helion.pl/przyklady/tenflo.zip

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
http://helion.pl/user/opinie/tenflo
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

Spis treści

3

ROZDZIAŁ

Analiza wydźwięku w przeglądarce przy użyciu TensorFlow.js

Analiza wydźwięku, zwana również analizą sentymentu (ang. *sentiment analysis*), to popularny problem w uczeniu maszynowym. Ludzie ciągle starają się zrozumieć wydźwięk opisu produktu lub recenzji filmowej. Obecnie w celu analizy wydźwięku pobieramy tekst z klienta lub przeglądarki i przekazujemy go do serwera, który uruchamia model uczenia maszynowego, aby przewidzieć wydźwięk tekstu, a następnie serwer odsyła wynik do klienta.

Nie ma problemu, jeśli nie przejmujemy się opóźnieniami w systemie. Istnieje jednak wiele zastosowań, takich jak handel akcjami czy rozmowy związane z obsługą klienta, w których przewidywanie wydźwięku z niskim opóźnieniem może być pomocne. Jednym z oczywistych wąskich gardeł w zmniejszaniu opóźnień jest wywoływanie serwera.

Jeśli analizę wydźwięku można byłoby uzyskać przy użyciu samej przeglądarki lub klienta, moglibyśmy pozbyć się wywołania do serwera i przewidywać wydźwięk w czasie rzeczywistym. Google udostępnił niedawno bibliotekę TensorFlow.js, która pozwala na przeprowadzenie szkolenia modelu i wnioskowanie w przeglądarce lub kliencie.

Dodatkowo szkolenie modeli po stronie klienta otwiera szereg możliwości. Poniżej znajduje się krótkie podsumowanie wszystkich zalet takiego działania:

- **prywatność** — ponieważ dane znajdują się wyłącznie po stronie klienta, jesteśmy w stanie zapewnić magiczne doświadczenie uczenia maszynowego bez narażania prywatności danych,
- **prostota** — ponieważ kod działa w przeglądarce, użytkownik nie musi instalować żadnych bibliotek ani zależności,
- **szybkość działania** — ponieważ nie ma potrzeby transferowania danych do serwerów w celu szkolenia lub prognozowania, możemy wdrażać modele uczenia maszynowego dla zastosowań wymagających krótkiego czasu oczekiwania,
- **brak zależności od urządzenia** — strona internetowa może być otwierana na dowolnym urządzeniu (laptopie, smartfonie itd.), dlatego TensorFlow.js może wykorzystywać dowolny sprzęt (GPU) lub czujniki urządzenia, takie jak akcelerometry w urządzeniach mobilnych, do szkolenia modeli uczenia maszynowego.

Pokażemy, jak włączyć analizę wydźwięku w przeglądarce za pomocą TensorFlow.js, omawiając następujące tematy:

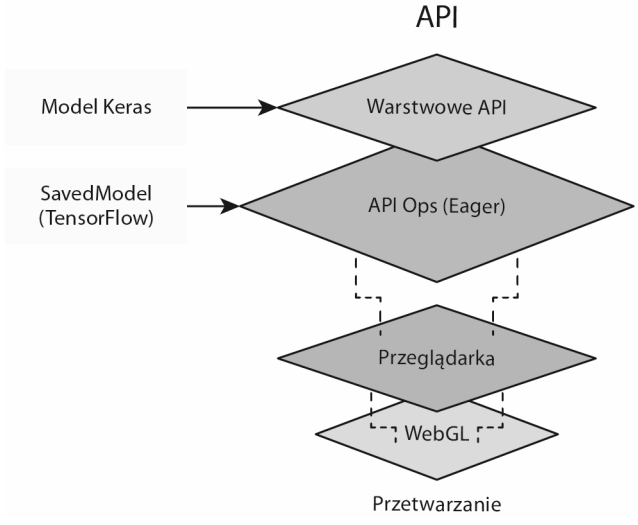
- TensorFlow.js,
- optymalizacja Adam,
- strata kategoryzacyjnej entropii krzyżowej,
- osadzanie słów,
- konfigurowanie problemu analizy wydźwięku i budowanie modelu w Pythonie,
- wdrażanie modelu w przeglądarce przy użyciu TensorFlow.js.

> Znajdź kod do tego rozdziału; instrukcje instalacji znajdują się również w pliku *README* w repozytorium dla tego projektu.

TensorFlow.js

TensorFlow niedawno udostępnił TensorFlow.js na licencji open source. Jest to biblioteka otwartoźródłowa, która pomaga nam definiować i szkolić modele uczenia głębokiego w całości w przeglądarce przy użyciu JavaScript, jak również poprzez wysokopoziomowe, warstwowe API. Możemy to wykorzystać do szkolenia modeli uczenia głębokiego wyłącznie po stronie klienta. Do szkolenia tych modeli nie jest wymagany serwer z układem GPU.

Poniższy diagram przedstawia zarys działania API TensorFlow.js:



Działa to w oparciu o **WebGL**. TensorFlow.js dostarcza również wysokopoziomowe, warstwowe API. Posiada też wsparcie dla wykonywania w trybie **Eager**. Przy użyciu TensorFlow.js można osiągnąć trzy rzeczy:

- wczytywać istniejące modele TensorFlow lub Keras w celu predykcji w przeglądarce,
- ponownie szkolić istniejące modele przy użyciu danych klienta,
- definiować i szkolić modele uczenia głębokiego od podstaw w przeglądarce.

Optymalizacja Adam

Zanim przyjrzymy się optymalizacji Adam, spróbujmy najpierw zrozumieć koncepcję gradientu prostego.

Metoda gradientu prostego jest iteracyjnym algorytmem optymalizacji w celu znalezienia minimum funkcji. Posłużmy się następującą analogią: powiedzmy, że utknęliśmy gdzieś na zboczu góry i chcemy jak najszybciej zejść na dół. Pierwszym krokiem będzie obserwowanie zbocza góry we wszystkich kierunkach wokół nas i podjęcie decyzji o podążaniu w kierunku największego nachylenia zbocza w dół.

Po każdym kroku ponownie oceniamy nasz wybór kierunku. Długość naszego spaceru zależy również od stopnia nachylenia zbocza w dół. Jeśli nachylenie jest bardzo strome, robimy większe kroki, ponieważ może to pomóc nam szybciej dotrzeć na dół. W ten sposób po pokonaniu mniejszej lub większej liczby kroków możemy bezpiecznie zejść na dół. Podobna sytuacja ma miejsce w uczeniu maszynowym, gdzie chcemy zminimalizować pewne błędy

i koszty poprzez aktualizację wag algorytmu. Aby znaleźć minimum funkcji kosztu przy użyciu gradientu, aktualizujemy wagi algorytmu proporcjonalnie do gradientu w kierunku najbardziej stromych zejść. W literaturze dotyczącej sieci neuronowych stała proporcjonalności znana jest również jako współczynnik uczenia.

W uczeniu maszynowym na dużą skalę optymalizacja metodą gradientu prostego jest jednak dość kosztowna, ponieważ robimy tylko jeden krok po pojedynczym przejściu na całym zbiorze danych treningowych. Tak więc przy kilku tysiącach kroków czas potrzebny do osiągnięcia minimum funkcji kosztu jest ogromny.

Rozwiązaniem tego problemu jest stochastyczny spadek wzdłuż gradientu (SGD — ang. *Stochastic Gradient Descent*), podejście, w którym aktualizacja wag algorytmu następuje po każdym przykładzie treningowym, bez czekania, aż cały zestaw danych przejdzie przez algorytm. Używamy terminu **stochastyczny**, aby zaznaczyć przybliżony charakter gradientu, ponieważ jest on obliczany po każdym przykładzie treningowym. Jednak z literatury wynika, że po wielu iteracjach SGD prawie na pewno zbiegnie się z prawdziwym lokalnym lub globalnym minimum funkcji. Ogólnie rzecz biorąc, w algorytmach uczenia głębokiego możemy zaobserwować tendencję do używania minigrup SGD, gdzie wagi aktualizowane są po każdej minigrupie, a nie po każdym przykładzie treningowym.

Optymalizacja Adam jest wariantem SGD, w którym utrzymujemy wskaźnik uczenia na parametr (wagę) i aktualizujemy go na podstawie średniej i wariancji poprzednich gradientów tego parametru. Optymalizacja Adam okazała się niezwykle dobra i szybka w rozwiązywaniu wielu problemów uczenia głębokiego. Więcej informacji na temat tej optymalizacji można znaleźć w oryginalnym artykule (*https://arxiv.org/abs/1412.6980*).

Strata kategoryzacyjnej entropii krzyżowej

Strata entropii krzyżowej (ang. *cross entropy loss*), inaczej strata logistyczna (ang. *log loss*), mierzy wydajność modelu klasyfikacji, którego wynik jest prawdopodobieństwem z przedziału od 0 do 1. Entropia krzyżowa wzrasta, w miarę jak przewidywane prawdopodobieństwo próbki odbiega od wartości rzeczywistej. Dlatego też przewidywanie prawdopodobieństwa na poziomie 0,05, podczas gdy rzeczywista etykieta ma wartość 1, zwiększa stratę entropii krzyżowej.

Z matematycznego punktu widzenia dla klasyfikacji binarnej entropia krzyżowa jest definiowana w następujący sposób:

$$-\left(y_i \log\left(p_i\right)+\left(1-y_i\right)\log\left(1-p_i\right)\right)$$

Tutaj y_i jest wskaźnikiem binarnym (0 lub 1) określającym klasę dla próbki i, podczas gdy p_i oznacza przewidywane prawdopodobieństwo z przedziału od 0 do 1 dla tej próbki.

W przeciwnym razie, jeśli są co najmniej trzy klasy, definiujemy nowy termin znany jako kategoryzacyjna entropia krzyżowa. Jest ona obliczana jako suma oddzielnych strat dla każdej etykiety klas na obserwację. Z matematycznego punktu widzenia jest to następujące równanie:

$$-\sum_{c=1}^{M} y_{i,c} \log(p_i, c)$$

Tutaj M oznacza liczbę klas, $y_{i,c}$ jest wskaźnikiem binarnym (0 lub 1), który określa, czy c jest właściwą klasą dla obserwacji i, a $p_{i,c}$ oznacza prawdopodobieństwo obserwacji i dla klasy c.

Ponieważ w tym rozdziale przeprowadzamy klasyfikację binarną na recenzjach, będziemy stosować tylko binarną entropię krzyżową jako naszą stratę klasyfikacyjną.

Osadzanie słów

Osadzanie słów odnosi się do klasy technik uczenia się cech w **przetwarzaniu języka naturalnego** (NLP — ang. *Natural Language Processing*), które są wykorzystywane do generowania rzeczywistej reprezentacji wektorowej słowa, zdania lub dokumentu.

Wiele zadań związanych z uczeniem maszynowym wiąże się dziś z tekstem. Dla przykładu, tłumaczenia językowe Google lub wykrywanie spamu w usłudze Gmail wykorzystują tekst jako dane wejściowe do swoich modeli w celu wykonania zadań tłumaczenia i wykrywania spamu. Jednak współczesne komputery mogą przyjmować jako dane wejściowe tylko liczby rzeczywiste i nie potrafią zrozumieć ciągów znaków lub tekstu, chyba że zakodujemy je w postaci liczb lub wektorów.

Rozważmy na przykład zdanie „Lubię piłkę nożną", dla którego chcemy oddać reprezentację wszystkich słów. Metoda *brute force* do wygenerowania osadzeń trzech słów „lubię", „piłkę" i „nożną" polega na przedstawieniu słów za pomocą kodowania z gorącą jedynką. W tym przypadku osadzanie wygląda następująco:

- „lubię" = [1,0,0],
- „piłkę" = [0,1,0],
- „nożną" = [0,0,1].

Idea polega na stworzeniu wektora, który ma wymiar równy liczbie unikatowych słów w zdaniu i przypisaniu jedynki do pozycji, w której słowo występuje, i zer wszędzie indziej. Z tym podejściem wiążą się dwie kwestie:

- Liczba wymiarów wektorowych skaluje się zgodnie z liczbą słów w korpusie. Powiedzmy, że mamy 100 000 unikatowych słów w dokumencie. Tym samym każde słowo reprezentujemy wektorem o wymiarze 100 000. Zwiększa to ilość pamięci potrzebnej do reprezentowania słów, przez co nasz system staje się nieefektywny.

- Tego rodzaju reprezentacja z gorącą jedynką nie potrafi uchwycić podobieństwa między słowami. Załóżmy, że w zdaniu istnieją dwa słowa, „lubię" i „kocham". Wiemy, że „lubię" jest bardziej podobne do „kocham" niż do „piłka". Jednak w naszej obecnej reprezentacji z gorącą jedynką iloczyn skalarny dowolnych dwóch wektorów wynosi zero. Z matematycznego punktu widzenia iloczyn skalarny słów „lubię" i „piłkę" z naszego zdania jest reprezentowany w następujący sposób:

  $$\text{„lubię"} \times \text{„piłkę"} = \text{Transpozycja}([1,0,0]) \times [0,1,0] = 1 \times 0 + 0 \times 1 + 0 \times 0 = 0$$

  Jest to spowodowane tym, że każde słowo ma w wektorze oddzielną pozycję, w której występuje tylko jedynka.

Zarówno w przypadku wykrywania spamu, jak i problemów związanych z tłumaczeniami językowymi zrozumienie podobieństwa słów jest dość istotne. Z tego powodu istnieje kilka innych sposobów (zarówno nadzorowanych, jak i nienadzorowanych) osadzania słów, które można wykorzystać w uczeniu maszynowym.

> Więcej informacji na temat osadzania słów w TensorFlow można znaleźć w oficjalnym samouczku (*https://www.tensorflow.org/tutorials/representation/word2vec*).

Ten projekt wykorzystuje warstwę osadzania z Keras do odwzorowania słów w naszych recenzjach filmowych na rzeczywiste reprezentacje wektorowe. W tym projekcie nauczymy się reprezentacji wektorowej słów w sposób nadzorowany. Zasadniczo inicjujemy osadzanie słów losowo, a następnie wykorzystujemy propagację wsteczną w sieciach neuronowych do aktualizacji osadzania w taki sposób, aby zminimalizować całkowitą stratę sieci. Szkolenie w sposób nadzorowany pomaga w generowaniu osadzeń specyficznych dla danego zadania. Na przykład oczekujemy podobnej reprezentacji słów takich jak „niesamowity" i „świetny", ponieważ oba te słowa mają pozytywny wydźwięk. Po zakodowaniu słów w recenzjach filmowych możemy je wykorzystać jako wejście do naszych warstw sieci neuronowych.

Budowanie modelu analizy wydźwięku

W tym podrozdziale dowiemy się, jak zbudować od podstaw model analizy wydźwięku przy użyciu Keras. Wykorzystamy do tego celu dane analizy wydźwięku z Uniwersytetu Michigan, dostępne na stronie *https://www.kaggle.com/c/si650winter11/data*. Ten zbiór danych zawiera 7086 recenzji filmowych z etykietami. Etykieta `1` oznacza pozytywny wydźwięk, natomiast `0` oznacza wydźwięk negatywny. W repozytorium zbiór danych jest przechowywany w pliku o nazwie *sentiment.txt*.

Wstępne przetwarzanie danych

Po zainstalowaniu pakietów (wymienionych w pliku *requirements.txt* z kodem) wymaganych do uruchomienia tego projektu i wczytania danych kolejnym krokiem będzie wstępne przetworzenie danych:

1. Pierwszym krokiem jest pozyskanie listy tokenów (słów) z recenzji. Usuń wszelkie znaki interpunkcyjne i upewnij się, że wszystkie tokeny są zapisane małymi literami:

```
def get_processed_tokens(text):
'''
Pobranie listy tokenów z recenzji
'''
filtered_text = re.sub(r'[^a-zA-Z0-9\s]', '', text) #Usuwanie znaków
interpunkcyjnych
filtered_text = filtered_text.split()
filtered_text = [token.lower() for token in filtered_text]
return filtered_text
```

   Dla przykładu, jeśli mamy wejście `Ten film jest SUPER!!!!`, naszym wyjściem powinno być `ten film jest super`.

2. Utwórz słownik `token_idx`, który odwzorowuje tokeny na liczby całkowite w celu utworzenia osadzenia. Zwróć uwagę, że liczba unikatowych tokenów (słów) obecnych w słowniku może być bardzo duża, dlatego musimy odfiltrować te, których liczba wystąpień w zbiorze treningowym jest mniejsza niż ustalony próg (w kodzie domyślna wartość wynosi `5`). Jest to spowodowane tym, że trudno jest zauważyć jakikolwiek związek pomiędzy wydźwiękiem a słowami, które nie występują zbyt często w zbiorze danych:

```
def tokenize_text(data_text, min_frequency =5):
    '''
    Tokenizacja recenzji w zbiorze danych. Odfiltrowywanie rzadko występujących tokenów.
    '''
    review_tokens = [get_processed_tokens(review) for review in
                     data_text] # Tokenizacja zdań
    token_list = [token for review in review_tokens for token in review]
    # Konwersja do pojedynczej listy
    token_freq_dict = {token:token_list.count(token) for token in
    set(token_list)} # Obliczanie częstotliwości występowania tokenów
    most_freq_tokens = [tokens for tokens in token_freq_dict if
    token_freq_dict[tokens] >= min_frequency]
    idx = range(len(most_freq_tokens))
    token_idx = dict(zip(most_freq_tokens, idx))
    return token_idx,len(most_freq_tokens)
```

3. Odwzoruj każdą recenzję w zbiorze danych do sekwencji liczb całkowitych (na podstawie słownika `token_idx`, który stworzyliśmy w poprzednim kroku). Jednak zanim to zrobisz, znajdź recenzję z największą liczbą tokenów:

```
def get_max(data):
    '''
    Obliczanie maksymalnej liczby tokenów na recenzję
    '''
    tokens_per_review = [len(txt.split()) for txt in data]
    return max(tokens_per_review)
```

4. Aby utworzyć sekwencje, które zostaną wprowadzone do modelu w celu nauki osadzeń, musimy utworzyć sekwencję o ustalonej długości (`max_tokens`) dla każdej recenzji w zbiorze danych. Aby zapewnić taką samą długość wszystkich sekwencji, wstępnie uzupełniamy sekwencje zerami, jeśli ich długość jest mniejsza niż maksymalna. Wstępne uzupełnianie sekwencji jest korzystniejsze niż późniejsze uzupełnianie, ponieważ pomaga osiągnąć dokładniejsze wyniki:

```
def create_sequences(data_text,token_idx,max_tokens):
    '''
    Tworzenie sekwencji odpowiednich dla wejścia GRU
    Wejście: dane recenzji, słownik z tokenami, max_tokens
    Wyjście: padded_sequences o kształcie (len(data_text), max_tokens)
    '''
    review_tokens = [get_processed_tokens(review) for review in
                     data_text] # Tokenizacja zdań
    # Konwersja tokenów na ich indeksy
    review_token_idx = map(lambda review: [token_idx[k] for k in review
                           if k in token_idx.keys() ],
    review_tokens)
        padded_sequences =
    pad_sequences(review_token_idx,maxlen=max_tokens)
        return np.array(padded_sequences)
```

Budowanie modelu

Model ten będzie się składał z warstwy osadzającej, a następnie trzech warstw GRU i w pełni połączonej warstwy z aktywacją sigmoidalną. Do pomiaru optymalizacji i dokładności wykorzystamy odpowiednio optymalizator Adam i `binary_crossentropy`:

1. Model jest definiowany przy użyciu następujących parametrów:

```
def define_model(num_tokens,max_tokens):
    '''
    Określanie definicji modelu na podstawie parametrów wejściowych
    '''
    model = Sequential()
    model.add(Embedding(input_dim=num_tokens,
                    output_dim=EMBEDDING_SIZE,
                    input_length=max_tokens,
                    name='layer_embedding'))

    model.add(GRU(units=16, name = "gru_1",return_sequences=True))
    model.add(GRU(units=8, name = "gru_2",return_sequences=True))
    model.add(GRU(units=4, name= "gru_3"))
    model.add(Dense(1, activation='sigmoid',name="dense_1"))
    optimizer = Adam(lr=1e-3)
    model.compile(loss='binary_crossentropy',
                  optimizer=optimizer,
                  metrics=['accuracy'])
    print model.summary()
    return model
```

2. Przeprowadź szkolenie modelu z następującymi parametrami:
   - liczba epok = 15,
   - podział walidacji = 0,05,
   - wielkość grupy = 32,
   - rozmiar osadzania = 8.

```
def train_model(model,input_sequences,y_train):
    '''
    Szkolenie modelu na podstawie parametrów wejściowych
    '''
    model.fit(input_sequences, y_train,
          validation_split=VAL_SPLIT, epochs=EPOCHS,
          batch_size=BATCH_SIZE)
    return model
```

3. Przetestuj model przeszkolony na kilku losowo wybranych zdaniach, aby zweryfikować jego działanie:

Tekst	Przewidywany wynik
Awesome movie	0,9957
Terrible Movie	0,0023
That movie really sucks	0,0021
I like that movie	0,9469

Przewidywany wynik jest bliski 1 dla zdań pozytywnych i bliski 0 dla zdań negatywnych. Losowe testy potwierdzają wydajność naszego modelu.

> Zwróć uwagę, że rzeczywiste wyniki mogą być nieco inne, jeśli szkolisz swój model na innego rodzaju sprzęcie.

Uruchamianie modelu w przeglądarce przy użyciu TensorFlow.js

W tym podrozdziale zamierzamy wdrożyć model w przeglądarce.

Poniższe kroki przedstawiają, jak zapisać model:

1. Zainstaluj bibliotekę TensorFlow.js, która pomoże nam sformatować nasz przeszkolony model tak, aby mógł zostać wykorzystany przez przeglądarkę:

```
pip install tensorflowjs
```

2. Zapisz model w formacie TensorFlow.js:

```
import tensorflowjs as tfjs
tfjs.converters.save_keras_model(model, OUTPUT_DIR)
```

W ten sposób zostanie utworzony plik json o nazwie *model.json*, który będzie zawierał metazmienne, a także inne pliki, takie jak *group1-shard1of1*.

Dobra robota! Wdrożenie modelu w pliku HTML jest jednak nieco trudniejsze:

> Aby uruchomić kod wymieniony w repozytorium, należy dokładnie zapoznać się z dokumentacją *README.md* (w razie potrzeby zwróć uwagę na część poświęconą rozwiązywaniu problemów), dotyczącą ustawień przed uruchomieniem pliku *Run_On_Browser.html*.

1. Włącz TensorFlow.js do swojego kodu JavaScript za pomocą znaczników `script`:

```
<script src="https://cdn.jsdelivr.net/npm/@tensorflow/tfjs@0.8.0"></script>
```

2. Załaduj model i nasz słownik `token_idx`. Pomoże nam to wczytać stosowne dane przed przetworzeniem jakiejkolwiek recenzji z przeglądarki:

```
async function createModel()
{
const model = await
tf.loadModel('http://127.0.0.1:8000/model.json')
return model
}
async function loadDict()
{
  await $.ajax({
  url: 'http://127.0.0.1:8000/token_index.csv',
  dataType: 'text',
  crossDomain : true}).done(success);
}
function success(data)
{
    var wd_idx = new Object();
    lst = data.split(/\r?\n|\r/)
    for(var i = 0 ; i < lst.length ;i++){
        key = (lst[i]).split(',')[0]
        value = (lst[i]).split(',')[1]
        if(key == "")
            continue
        wd_idx[key] = parseInt(value)
    }
    word_index = wd_idx
}

async function init()
{
  word_index = undefined
  console.log('Rozpoczęcie wczytywania słownika')
```

```
  await loadDict()
  //console.log(word_index)
  console.log('Koniec wczytywania słownika')
  console.log('Rozpoczęcie wczytywania modelu')
  model = await createModel()
  console.log('Koniec wczytywania modelu')
}
```

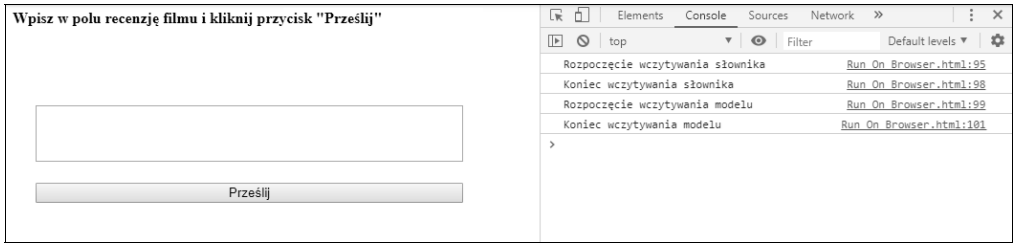
3. Dodaj kilka funkcji pomocniczych do przetwarzania danych wejściowych recenzji z przeglądarki. Obejmuje to przetwarzanie tekstu, odwzorowywanie słów do `token_idx` oraz tworzenie sekwencji dla predykcji modelu:

```
function process(txt)
{
 out = txt.replace(/[^a-zA-Z0-9\s]/, '')
 out = out.trim().split(/\s+/)
 for (var i = 0 ; i < out.length ; i++)
 out[i] = out[i].toLowerCase()
 return out
}

function create_sequences(txt)
{
 max_tokens = 40
 tokens = []
 words = process(txt)
 seq = Array.from(Array(max_tokens), () => 0)
 start = max_tokens-words.length
 for(var i= 0 ; i< words.length ; i++)
 {
     if (Object.keys(word_index).includes(words[i])){
         seq[i+start] = word_index[words[i]]
     }
 }
 return seq
}
```

4. Dołącz funkcję predykcji, która przetwarza zdanie wejściowe i wykorzystuje funkcję predykcji modelu, aby zwrócić tensor z przewidywanym wynikiem, jak pokazano w poprzednim punkcie:

```
async function predict()
{
 txt = document.getElementById("userInput").value
 alert(txt);
 seq = create_sequences(txt)
 input = tf.tensor(seq)
 input = input.expandDims(0)
 pred = model.predict(input)
 document.getElementById("Sentiment").innerHTML = pred;

 pred.print()
}
```

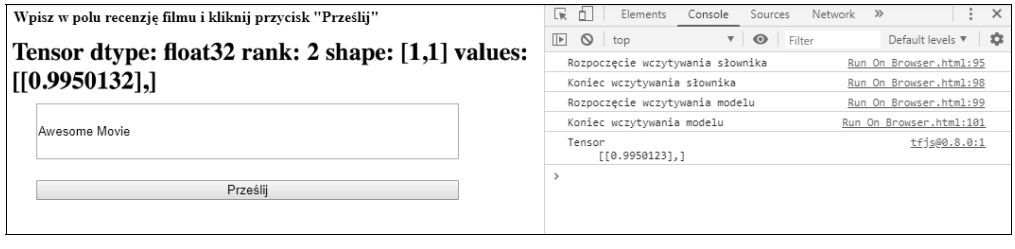
5. Aby zilustrować cały proces z punktu widzenia użytkownika, otwórz plik *Run_on_Browser.html*. Zobaczysz coś podobnego do tego, co jest pokazane na poniższym zrzucie ekranu:



Lewa strona zrzutu ekranu przedstawia układ strony internetowej, podczas gdy prawa strona pokazuje konsolę i wyjścia.

> Zauważ, że słownik i model możemy załadować wcześniej, aby przyspieszyć predykcje.

6. Wpisz recenzję w odpowiednie pole i kliknij przycisk *Prześlij*, aby zobaczyć przewidywany wynik modelu. Spróbuj uruchomić aplikację z tekstem `Awesome Movie`:



Przewidywany wynik jest dość wysoki, co wskazuje na pozytywny wydźwięk. Możesz poeksperymentować z różnymi tekstami, aby sprawdzić wyniki.

> Zwróć uwagę, że służy to głównie celom ilustracyjnym. Jeśli chcesz, możesz poprawić interfejs użytkownika poprzez JavaScript.

Podsumowanie

Ten rozdział był krótkim wprowadzeniem do budowania kompletnego systemu, który szkoli model analizy wydźwięku przy użyciu Keras i wdraża go w JavaScript przy użyciu TensorFlow.js. Proces wdrażania modelu do produkcji jest dość bezproblemowy.

Potencjalnym kolejnym krokiem byłoby zmodyfikowanie kodu JavaScript, tak aby przewidywanie wydźwięku następowało natychmiast po wpisaniu słowa. Jak wspomnieliśmy wcześniej, wdrożenie modelu przy użyciu TensorFlow.js umożliwia wykorzystanie go w zastosowaniach wymagających niskich opóźnień, takich jak przewidywanie wydźwięku w czasie rzeczywistym bez konieczności interakcji z serwerem.

W końcu zbudowaliśmy sieć neuronową w Pythonie i wdrożyliśmy ją w JavaScript. Możesz jednak spróbować zbudować cały model w JavaScript, używając TensorFlow.js.

W następnym rozdziale poznamy nową bibliotekę Google, TensorFlow Lite.

Kwestie do rozważenia

1. Czy możesz ocenić dokładność modelu, jeśli używasz LSTM zamiast GRU?
2. Co dzieje się z dokładnością i czasem szkolenia w przypadku zwiększenia rozmiaru osadzania?
3. Czy możesz dodać więcej warstw do modelu? Jak to wpłynie na czas szkolenia?
4. Czy możesz zmodyfikować kod, aby szkolić model w przeglądarce, zamiast wczytywać wyszkolony model?

TensorFlow: prostota, wydajność i imponujący potencjał!

TensorFlow służy do projektowania i wdrażania zaawansowanych architektur głębokiego uczenia. Jego zaletami są prostota, wydajność i elastyczność. Umożliwia budowanie złożonych rozwiązań na bazie różnorodnych zbiorów danych. Co więcej, pozwala na stosowanie różnych technik uczenia nadzorowanego, nienadzorowanego oraz uczenia przez wzmacnianie. TensorFlow zmienił sposób postrzegania uczenia maszynowego. Dzięki temu środowisku każdy, kto chce uczynić z dużych zbiorów danych wiarygodne źródło wiedzy, może ten cel osiągnąć — niezależnie od tego, czy jest analitykiem danych, naukowcem, projektantem, czy pasjonatem metod sztucznej inteligencji.

To książka przeznaczona dla osób, które chcą nauczyć się tworzyć całościowe rozwiązania z wykorzystaniem uczenia maszynowego. Poszczególne zagadnienia zilustrowano trzynastoma praktycznymi projektami, w których wykorzystano między innymi analizy sentymentów, przetwarzanie języka naturalnego, systemy rekomendacyjne, generatywne sieci kontradyktoryjne czy sieci kapsułowe. Pokazano, w jaki sposób używać TensorFlow z interfejsem APO Spark i wspomagać obliczenia układami GPU. Przedstawiono zastosowanie rozkładu macierzy (SVD++), modeli rankingowych i odmian splotowej sieci neuronowej. Nie zabrakło prezentacji nowych rozwiązań o dużym potencjale, takich jak sieci DiscoGAN. Dołączony do książki kod źródłowy, liczne wskazówki i porady pozwolą na płynne rozpoczęcie pracy z TensorFlow oraz innymi narzędziami do budowy sieci neuronowych.

W tej książce między innymi:

- podstawy pracy z TensorFlow
- wykorzystanie TensorFlow do wizualizacji sieci neuronowych
- zastosowanie procesu gaussowskiego do prognozowania cen akcji
- wykrywanie oszukańczych transakcji za pomocą TensorFlow i Keras
- implementacja sieci kapsułowych w TensorFlow

Ankit Jain — jest naukowcem. Pracuje w oddziale badawczym Ubera, gdzie zajmuje się metodami głębokiego uczenia. Wcześniej wykładał na Uniwersytecie Kalifornijskim w Berkeley.

Armando Fandango — specjalizuje się w dziedzinie głębokiego uczenia, uczenia maszynowego, rozproszonego przetwarzania danych i metod obliczeniowych. Jest konsultantem, projektantem i autorem książek.

Amita Kapoor — od dwudziestu lat wykłada wiedzę o sieciach neuronowych na Uniwersytecie w Delhi. Interesuje się uczeniem maszynowym, sieciami neuronowymi, robotyką oraz buddyzmem i etyką w sztucznej inteligencji.

ISBN 978-83-283-5708-2
9 788328 357082
Cena: 59,00 zł

Packt