Projekt kompetencyjny

Rekurencyjne sieci neuronowe

	Autor	**Album**
	Kateryna Tsarova	226451

	Prowadzący
	Piotr Duch, dr inż.

Ocena prowadzącego:

Łódź, 10.02.2021 r.

SPIS TREŚCI

1. WSTĘP

1.1 Cel projektu

Celem mojego projektu jest stworzenie sieci LSTM, umożliwiającej generowanie tekstu.

1.2 Czym są rekurencyjne sieci neuronowe

Rekurencyjne sieci neuronowe (ang. *recurrent neural networks, RNN*) to rodzina sieci neuronowych służąca do pracy z danymi sekwencyjnymi. Sieci rekurencyjne mają zastosowanie w tłumaczeniu tekstu, rozpoznawaniu obrazu, rozpoznawaniu mowy czy generowaniu muzyki, ponieważ wszystkie te mechanizmy potrzebują do działania sekwencji. Jedną z pierwszych koncepcji systemów uczących się i modeli statystycznych była możliwość współdzielenia parametrów w różnych częściach modelu. Takie podejście pozwala na rozszerzenie modelu i zastosowanie go do przykładów o różnych postaciach. Istnieje klika różnych kategorii przetwarzania danych sekwencyjnych, które można podzielić ze względu na dane wejściowe i wyjściowe.

Można je podzielić na trzy główne kategorie:

- Wiele do jednego – Dane wejściowe to sekwencja, wyjście to wektor o stałym rozmiarze. Dla przykładu na wejściu jest fragment tekstu, a na wyjściu jedno słowo np. emocja opisująca fragment tekstu, inny przykład to sekwencja nut na wejściu, a na wyjściu nazwa gatunku muzycznego pasującego do zadanej sekwencji.
- Jeden do wielu – Dane wejściowe nie są sekwencją, natomiast dane wyjściowe składają się z sekwencji. Przykład to opisywanie obrazów, wejście to obraz, a na wyjściu opis obrazu.
- Wiele do wielu – Dane wejściowe i wyjściowe składają się z sekwencji. Przykład to tłumaczenie jednego języka na inny, generowanie nowego utworu muzycznego na podstawie innego.

Sieci LSTM zostały zaprojektowane w celu uniknięcia problemu długotrwałej zależności. Trening sieci LSTM pozwala na zapamiętanie długotrwałych zależności pomiędzy danymi wejściowymi, a wyjściowymi wynikami zwracanymi przez sieć. Struktura komórki LSTM może przypominać pamięć komputera, ponieważ może odczytywać, zapisywać i usuwać informacje za pomocą odpowiednich bramek. Komórka decyduje, czy przechować lub usunąć informacje w zależności od przypisanej wagi do informacji [1].

1.3 Poznane techniki i narzędzia

Do napisania sieci neuronowej użyłam języka Python, przy napisaniu korzystałam z bibliotek Tensorflow, Keras.

1.4 Przebieg pracy

Projekt polega na zaimplementowanie rekurencyjnej sieci neuronowej, która może generować własne cytaty.

2. DZIAŁANIE REKURENCYJNYCH SIECI NEURONOWYCH

2.1 Przygotowanie danych wejściowych

Do wytrenowania sieci neuronowej była wykorzystana baza danych, która składa się z 36 165 cytatów (878 450 słów) pochodzących od 2297 znanych osób (autorzy, piosenkarze, politycy, sportowcy, naukowcy itp.) [2].

Proces przygotowania danych wejściowych wygląda następująco:

1) Z wczytanego zestawu danych jest tworzony słownik - tabela wszystkich unikalnych znaków, występujących w bazie danych,
2) Do każdego znaku w słowniku jest przypisana wartość z zakresu od 0 do długości słownika - 1,
3) Zestaw danych jest podzielony na zdania,
4) Proces przekształcania zdań w postać liczbową:
   - każdy znak w zdaniu jest zamieniony na wartość przypisaną do tego znaku (zob. pkt 2),

```
If you live to be a hundred, I want to live to be a hundred minus one day so I never have to live without you.

[29, 52, 1, 71, 61, 67, 1, 58, 55, 68, 51, 1, 66, 61, 1, 48, 51, 1, 47, 1, 54, 67, 60, 50, 64, 51, 50, 6, 1, 29, 1, 69, 47, 60,
66, 1, 66, 61, 1, 58, 55, 68, 51, 1, 66, 61, 1, 48, 51, 1, 47, 1, 54, 67, 60, 50, 64, 51, 50, 1, 59, 55, 60, 67, 65, 1, 61, 60,
51, 1, 50, 47, 71, 1, 65, 61, 1, 29, 1, 60, 51, 68, 51, 64, 1, 54, 47, 68, 51, 1, 66, 61, 1, 58, 55, 68, 51, 1, 69, 55, 66, 54,
61, 67, 66, 1, 71, 61, 67, 8, 0]
```

Rys. 1: Zdanie w postaci normalnej (góra) i w postaci liczbowej (dół)

5) Została zdeklarowana długość sekwencji, podawanej na wejście sieci (*seq_length*), a także *step* - o ile będzie przesuwany wskaźnik na początek następnej sekwencji:
   seq_length - 15,
   step - 6,
6) Proces rozbicia zdań na sekwencje wejściowe:
   - co każdy *step* znaków jest pobierana ze zdania następna sekwencja znaków o długości *seq_length,*

```
['If you live to ',
 ' live to be a h',
 'to be a hundred',
 'a hundred, I wa',
 'red, I want to ',
 ' want to live t',
 'to live to be a',
 'e to be a hundr',
 'e a hundred min',
 'ndred minus one',
 'minus one day s',
 'one day so I ne',
 'y so I never ha',
 ' never have to ',
 ' have to live w',
 'to live without',
 'e without you.\n']
```

Rys. 2: Zdanie, podzielone na sekwencje
(dla ułatwienia przedstawienia idei procesu rozbicia zdań na sekwencje zdanie jest pokazywane w postaci normalnej)

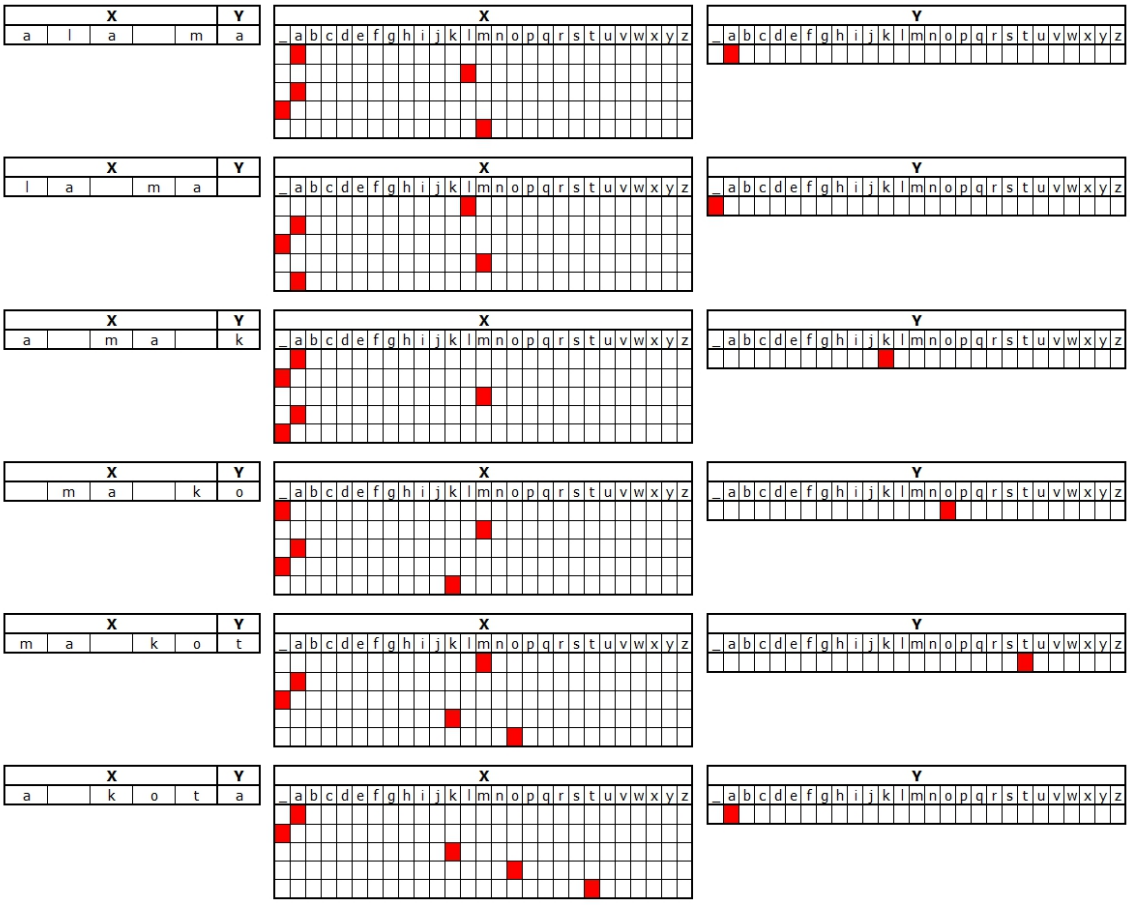
```
Sekwencja, podana na wejście sieci: If you live to
Oczekiwany wynik działania sieci: b

Sekwencja, podana na wejście sieci:  live to be a h
Oczekiwany wynik działania sieci: u

Sekwencja, podana na wejście sieci: to be a hundred
Oczekiwany wynik działania sieci: ,
```

Rys. 3: Przykłady danych wejściowych i oczekiwanych danych wyjściowych sieci

7) Każdy znak jest zapisywany jako one-hot vector:
   - każdy znak jest interpretowany jako wektor o długości *len(słownik)*, wypełniony zerami
   - na miejsce o indeksie, równym przyporządkowanej do znaku wartości (zob. pkt 2), jest wstawiona 1



Rys. 4: Przykład tablicy one-hot (X - dane wejściowe, Y - dane wyjściowe) [3]

8) Dane wejściowe i wyjściowe w postaci tabeli one-hot są podawane na wejściu sieci.

2.2 Model LSTM

Model rekurencyjnej sieci neuronowej był stworzony w *Tensorflow* (open-source'owy framework stworzony przez Google'a do obliczeń numerycznych). Oferuje on zestaw narzędzi służących do projektowania, trenowania oraz uczenia sieci neuronowych [4].

Każda warstwa modelu ma dokładnie jeden tensor wejściowy i jeden tensor wyjściowy - jest to model sekwencyjny (rys. 2). Sieć neuronowa składa się z kilku rodzajów warstw, oferowanych przez *Tensorflow*:

- *LSTM* – długa pamięć krótkotrwała,
- *Bidirectional LSTM* – dwukierunkowy LSTM,
- *Dense* – warstwa, dla której wszystkie jednostki poprzedniej warstwy są połączone ze wszystkimi w następnej,
- *Dropout* – przepuszczających tylko fragment danych (aby zapobiec przeuczeniu sieci).

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
bidirectional_1 (Bidirection (None, 15, 512)           688128
_________________________________________________________________
dropout_1 (Dropout)          (None, 15, 512)           0
_________________________________________________________________
lstm_2 (LSTM)                (None, 15, 128)           328192
_________________________________________________________________
dropout_2 (Dropout)          (None, 15, 128)           0
_________________________________________________________________
lstm_3 (LSTM)                (None, 64)                49408
_________________________________________________________________
dropout_3 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_1 (Dense)              (None, 1580)              102700
_________________________________________________________________
dropout_4 (Dropout)          (None, 1580)              0
_________________________________________________________________
dense_2 (Dense)              (None, 790)               1248990
_________________________________________________________________
dropout_5 (Dropout)          (None, 790)               0
_________________________________________________________________
dense_3 (Dense)              (None, 79)                62489
=================================================================
Total params: 2,479,907
Trainable params: 2,479,907
Non-trainable params: 0
```

Rys. 5: Architektura sieci neuronowej

Jako dane wejściowe warstwa wejściowa przyjmuje sekwencję liczbową o rozmiarze *seq_length.* Na wyjściu sieć generuje wektor one-hot.

Liczba neuronów w trzech warstwach LSTM jest równa odpowiednio 256, 128 i 64, liczba neuronów w trzech warstwach *Dense* – 20 * *len(słownik)*, 10 * *len(słownik)*, *len(słownik)*. Parametr warstwy *Dropout* jest równy 0.1. Warstwa wyjściowa korzysta z funkcji optymalizacji *softmax*, dzięki czemu można otrzymać na wyjściu tablicę składającą się z *len(słownik)* wartości sumujących się do 1.

Aby ocenić wydajność sieci neuronowej, do wytrenowania modelu była wykorzystana kategoryczna entropia krzyżowa (ang. *categorical crossentropy*). Był także wykorzystany algorytm optymalizacji ADAM (*ang. adaptive moment estimation*).

2.3 Wyniki nauczania

Uczenie modelu było dokonywane trzyetapowo. Do trenowania modelu były wybrane takie parametry:

1 etap:
- *epoch* – 15
- *batch_size* – 128

2 etap:
- *epoch* – 2
- *batch_size* – 2048

3 etap:
- *epoch* – 2
- *batch_size* – 1024

Epoch odpowiada za liczbę epok użytych do trenowania modelu – jedna epoka oznacza przejście całego zbioru przez sieć. *Batch_size* odpowiada za zdefiniowanie ile rekordów (obserwacji) przechodzi na raz podczas pojedynczego przebiegu zanim nastąpi pierwsza aktualizacja wag parametrów.

Końcowa dokładność sieci neuronowej jest równa 67.6%. Czas treningu - 4.5 godz.

Proces nauczania można zaobserwować na rys. 3.

```
Epoch 1/15
753133/753133 [==============================] - 912s 1ms/step - loss: 1.8891 - accuracy: 0.4459
Epoch 2/15
753133/753133 [==============================] - 999s 1ms/step - loss: 1.5231 - accuracy: 0.5427
Epoch 3/15
753133/753133 [==============================] - 907s 1ms/step - loss: 1.4160 - accuracy: 0.5718
Epoch 4/15
753133/753133 [==============================] - 920s 1ms/step - loss: 1.3538 - accuracy: 0.5892
Epoch 5/15
753133/753133 [==============================] - 955s 1ms/step - loss: 1.3115 - accuracy: 0.6007
Epoch 6/15
753133/753133 [==============================] - 944s 1ms/step - loss: 1.2808 - accuracy: 0.6089
Epoch 7/15
753133/753133 [==============================] - 940s 1ms/step - loss: 1.2559 - accuracy: 0.6154
Epoch 8/15
753133/753133 [==============================] - 914s 1ms/step - loss: 1.2348 - accuracy: 0.6211
Epoch 9/15
753133/753133 [==============================] - 943s 1ms/step - loss: 1.2167 - accuracy: 0.6257
Epoch 10/15
753133/753133 [==============================] - 1029s 1ms/step - loss: 1.2005 - accuracy: 0.6302
Epoch 11/15
753133/753133 [==============================] - 2094s 3ms/step - loss: 1.1862 - accuracy: 0.6338
Epoch 12/15
753133/753133 [==============================] - 994s 1ms/step - loss: 1.1719 - accuracy: 0.63720s
Epoch 13/15
753133/753133 [==============================] - 1001s 1ms/step - loss: 1.1602 - accuracy: 0.6413
Epoch 14/15
753133/753133 [==============================] - 910s 1ms/step - loss: 1.1485 - accuracy: 0.6439
Epoch 15/15
753133/753133 [==============================] - 910s 1ms/step - loss: 1.1382 - accuracy: 0.6469
Epoch 1/2
753133/753133 [==============================] - 346s 459us/step - loss: 1.0611 - accuracy: 0.6680
Epoch 2/2
753133/753133 [==============================] - 353s 469us/step - loss: 1.0378 - accuracy: 0.6746
Epoch 1/2
753133/753133 [==============================] - 391s 519us/step - loss: 1.0358 - accuracy: 0.6751
Epoch 2/2
753133/753133 [==============================] - 385s 511us/step - loss: 1.0293 - accuracy: 0.6764
```

Rys. 6: Proces nauczania sieci

Na Rys. 6 można zobaczyć, jak wraz z kolejnymi epokami wygląda metryka *accuracy* i *loss* sieci. Widoczne, że dokładność wzrasta w ciągu całego procesu nauczania sieci (straty odpowiednio zmniejszają się), zatem teoretycznie można byłoby zwiększyć liczbę epok dla każdego z etapów nauczenia sieci.

Zwiększenie liczby epok zwykle zapewnia wyższą dokładność, ale czas obliczeń również znacznie się wydłuża. Do osiągnięcia zamierzonego celu wystarczyły 3 etapy (15, 2, 2 epoki odpowiednio), ponieważ poza tym nie ma prawie żadnego wzrostu dokładności sieci, ale jej trening zajmuje o kilka razy więcej czasu.

2.4 Wyniki działania sieci

Wyniki działania sieci można zaobserwować na rys. 3.

Input	Output
I	Internet is understanding. I was better. Irish.
I am	I am. I am in my life. I am part of you.
You	You can't see what i have done. You have an elevolation, the part of self-insight and self. You know that is not funny thing.
This	This is something that makes me talk. This modern man is a man. This is an interesting enough.

Rys. 7: Wyniki działania sieci

Jest widoczne, że w większości wypadków sieć generuje poprawne zdania pod względem gramatycznym i ortograficznym, ale nie zawsze generowane zdania są poprawne pod względem logicznym. Jest to szczególnie wyraźne przy generacji dłuższych zdań, co można zobaczyć na następnym przykładzie.

Church was always enough for me and really change the economy for creative attitude, not already be your character of a horror.

Niniejsze zdanie dobrze ilustruje zalety i wady danego podejścia do nauczenia sieci: sieć nie popełnia błędów ortograficznych i gramatycznych i w większości wypadków poprawnie łączy słowa w frazy, które mają sens w kontekście lokalnym, ale końcowe zdanie często nie jest sensowne, i im dłuższe jest zdanie, tym większe ono ma problemy z logiczną spójnością.

Mimo to, w przeważającej większości wypadków wytrenowana sieć neuronowa spełnia swoje zadanie i jest w stanie generować gramatycznie poprawne zdania.

3. PODSUMOWANIE

Dzięki realizacji tego projektu zdobyłam szeroki zakres wiedzy i umiejętności w dziedzinach takich jak:

- pisanie w języku Python,
- zapoznanie z uczeniem maszynowym,
- zapoznanie z uczeniem głębokim,
- zapoznanie z działaniem rekurencyjnych sieci neuronowych,
- rozwiązywanie problemów z dziedziny informatyki,
- organizacja pracy.

5. ŹRÓDŁA

[1]https://blog.lukaszogan.com/informatyka/rekurencyjne-sieci-neuronowe-generuja-muzyke/
[2]https://www.kaggle.com/alvations/quotables/data
[3]https://blog.prokulski.science/index.php/2018/03/05/lstm-przewidywanie-tekstu-keras-r/
[4]https://geek.justjoin.it/zalety-wady-korzystania-tensorflow-srodowisku-produkcyjnym