

Raport z projektu

Przedmiot: Przetwarzanie języka naturalnego

Temat: Generator tytułów publikacji naukowych

Twórcy:

Piotr Warcholek

Karol Wojtasz

Paweł Wypych

1. Wstęp

Projekt polegał na stworzeniu programu, który generuje tytuły publikacji naukowych. Do jego wykonania wykorzystaliśmy **csinva/gpt-papertitle-generator** znajdującego się na portalu github.

Jako dane służące do uczenia modelu użyliśmy **arXiv**. ArXiv jest elektronicznym archiwum naukowych preprintów, które udostępnia ponad 2 mln artykułów naukowych.

Zdecydowaliśmy się na dodanie funkcji generowania tytułów na podstawie słów kluczowych, dzięki czemu mamy większy wpływ na otrzymany wynik.

Projekt gpt-paper-title-generator korzysta z **GPT3**, czyli produktu firmy OpenAI. Wykorzystuje deep learning w celu tworzenia tekstu podobnego do ludzkiego. GPT-3 jest ogólnodostępny. Z racji na to, że korzystanie z API jest dość kosztowne zrezygnowaliśmy z wykorzystania go w projekcie. Zamiast **GPT3** postanowiliśmy użyć sieci neuronowych na bazie biblioteki **keras** model Sequential.

2. Program

Wczytanie pliku z danymi i przekształcenie go na potrzeby uczenia maszynowego:

```
File already exists
46      data: Intelligent location of simultaneously a...
49      data: Intelligent location of simultaneously a...
303     data: The World as Evolving Information
670     data: Learning from compressed observations
953     data: Sensor Networks with Random Links: Topol...
...
128578  data: Matrix Completion from Noisy Entries
128585  data: Regularization methods for learning inco...
128705  data: From formulas to cirquents in computabil...
128779  data: Characterising equilibrium logic and nes...
128825  data: A Neural Network Classifier of Volume Da...
Name: title, Length: 1000, dtype: object
```

Do przechowywania danych wykorzystywane są pliki PKL. Pliki te, generowane przez moduł „pickle” w Pythonie, umożliwiają serializację obiektów, a następnie deserializację do programu w czasie wykonywania. Zasadniczo plik PKL zawiera strumień bajtów reprezentujący przechowywane obiekty.

Po przygotowaniu za pomocą funkcji `get_metadata()` naszym celem jest pokazanie wyłącznie kolumny tytułowej.

Biorąc pod uwagę dużą liczbę wierszy w bazie danych Arxiv, zawężamy obszar zainteresowania uczeniem sieci neuronowej, wybierając tylko początkowe 1000 wierszy.

Wektoryzacja tekstu:

```
[40] def split_target(chunk):  
    return chunk[:-1], chunk[1:]  
  
    full_text = " ".join(data.tolist())  
  
    vocab = sorted(set(full_text))  
    char_to_index = {char: index for index, char in enumerate(vocab)}  
    sequences = tf.data.Dataset.from_tensor_slices(np.array([char_to_index[char] for char in full_text])).batch(101, drop_remainder=True)  
    dataset = sequences.map(split_target)  
    BUFFER_SIZE = 10000  
    BATCH_SIZE = 10  
    dataset = dataset.shuffle(BUFFER_SIZE).batch(BATCH_SIZE, drop_remainder=True)
```

Aby przekonwertować wektor tekstowy na strumień indeksów znakowych, używamy funkcji `tf. data. Dataset.`

`from_tensor_slices`. Funkcja `dataset. shuffle()`. `batch()` służy do zapewnienia, że model nie uczy się wzorców na podstawie kolejności przykładów w zbiorze danych. Batching może poprawić wydajność obliczeniową, ponieważ model uczy się na danych, a nie na poszczególnych punktach danych.

Utworzenie modelu do uczenia maszynowego, wykorzystując bibliotekę `keras` oraz `tensorflow`:

```
[31] def create_model(vocab_size, embedding_dim, rnn_units, batch_size):  
    return tf.keras.Sequential([  
        tf.keras.layers.Embedding(vocab_size, embedding_dim, batch_input_shape=[batch_size, None]),  
        tf.keras.layers.GRU(rnn_units, return_sequences=True, stateful=True, recurrent_initializer='glorot_uniform'),  
        tf.keras.layers.Dense(vocab_size)  
    ])  
    embedding_dim = 100  
    rnn_units = 100  
  
    model = create_model(vocab_size=len(vocab), embedding_dim=embedding_dim, rnn_units=rnn_units, batch_size=BATCH_SIZE)
```

Wykorzystana została funkcja

`sparse_categorical_crossentropy` będąca funkcją typu “loss” z biblioteki `keras`:

```
[32] def loss(labels, logits):  
    return tf.keras.losses.sparse_categorical_crossentropy(labels, logits, from_logits=True)  
  
    model.compile(optimizer='adam', loss=loss, run_eagerly=True)  
    checkpoint_prefix = os.path.join('./training_checkpoints', "ckpt_{epoch}")  
    checkpoint_callback = tf.keras.callbacks.ModelCheckpoint(filepath=checkpoint_prefix, save_weights_only=True)
```

Funkcja straty mierzy różnicę między przewidywanymi wartościami modelu a rzeczywistymi wartościami zmiennej docelowej. Celem szkolenia jest zminimalizowanie tej funkcji strat.

Uczenie modelu zostało przeprowadzone na 75 epokach, według testów minimalna wartość która zapewnia prawidłowe wyniki to 50 epok, natomiast przy użyciu więcej niż 200 model zostaje przeuczony:

```
[33] EPOCHS = 75
      history = model.fit(dataset, epochs=EPOCHS, callbacks=[checkpoint_callback])

63/63 [=====] - 2s 37ms/step - loss: 1.2067
Epoch 48/75
63/63 [=====] - 2s 37ms/step - loss: 1.1999
Epoch 49/75
63/63 [=====] - 3s 42ms/step - loss: 1.1962
Epoch 50/75
63/63 [=====] - 2s 36ms/step - loss: 1.1897
Epoch 51/75
63/63 [=====] - 2s 36ms/step - loss: 1.1824
```

Funkcja generująca tytuł na podstawie modelu:

```
changeCharToIndex = {u:i for i, u in enumerate(vocab)}
changeIndexToChar = np.array(vocab)

def gen_title(model, start_string):
    text_length = 50
    input_eval = [changeCharToIndex[s] for s in start_string]
    input_eval = tf.expand_dims(input_eval, 0)
    text_generated = []
    model.reset_states()
    i = 0
    while 1:
        i += 1
        temp = 0.05
        pred = tf.squeeze(model(input_eval), 0) / temp
        pred_id = tf.random.categorical(pred, num_samples=1)[-1,0].numpy()
        input_eval = tf.expand_dims([pred_id], 0)
        text_generated.append(changeIndexToChar[pred_id])
        if (i > text_length and changeIndexToChar[pred_id] == " "):
            break

    return (start_string + ''.join(text_generated))
```

Uzyskane wyniki dla podanych fraz początkowych:

✓
3 s

```
print(gen_title(model, start_string="Neural Network"))
print(gen_title(model, start_string="Statistics"))
print(gen_title(model, start_string="Machine Learning"))
```

Neural Network Sequences Finding and the Computational Structure
Statistics of a new generalization of the Computational Statistical
Machine Learning Computational Structure of Semantic Web a Computational

3. Podsumowanie

Wykonaliśmy program generujący tytuły publikacji naukowych na podstawie słów początkowych. Dzięki zastosowaniu sieci neuronowych nie musieliśmy używać GPT3, wpłynęło to również pozytywnie na uzyskane wyniki. Jeśli wiemy, co w rezultacie chcielibyśmy uzyskać, możemy spróbować wyczyścić zbiór danych i nieznacznie zmienić teksty, które mamy, możemy użyć znacznie większej liczby tekstów i znacznie potężniejszych modeli (na przykład zwiększając liczbę jednostek i warstw RNN). Na bazie poprzedniego stanu RNN oraz danych wejściowych możemy przewidywać klasę następnego znaku.

4. Bibliografia

- ☐ „Github,” [Online]. Available:
<https://github.com/csinva/gpt-paper-titlegenerator>.
- ☐ „Tensorflow,” [Online]. Available:
https://www.tensorflow.org/text/tutorials/text_generation.
- ☐ <https://arxiv.org/>
- ☐ <https://towardsdatascience.com/generating-scientific-papers-titles-using-machine-learning-98c8c9bc637e>
- ☐ https://keras.io/guides/sequential_model/