Rodzina Bert – szerzej

Tasekcja kontynuuje dyskusję na temat modelu BERT opartego na transformatorach, obejmując takie tematy, jak tokenizator specyficzny dla BERT, podobieństwo zdań, generowanie tokenów BERT oraz rodzina tokenów BERT LLM.Pierwsza część zawiera przykład kodu w Pythonie dla specjalnych tokenów BERT, a następnie omówienie tokenizatora BERT i tokenizatora DistilBERT. Ta sekcja pokazuje również, jak przeprowadzić analizę sentymentu za pomocą BERT. Druga część opisuje sekwencję kroków kodowania BERT. Następnie dowiesz się o podobieństwie zdań w BERT, a także o Sentence-BERT. Zobaczysz również, jak generować tokeny BERT. Trzecia część zawiera listę tokenów LLM należących do rodziny BERT, takich jak ALBERT, BART i DistilBERT. W tej sekcji znajduje się również krótki przykład kodu w Pythonie do trenowania modelu BERT na laptopie, a także informacje dotyczące RoBERTa LLM. Czwarta część tego rozdziału szereg tematów, takich jak tłumaczenie na język włoski i japoński, narzędzia internetowe dla BERT, modelowanie tematów za pomocą BERT oraz Google PaLM.

PRZYKŁAD KODU DLA SPECJALNYCH TOKENÓW BERT

BERT używa specjalnego tokena [CLS] do wskazania początku ciągu tekstowego oraz specjalnego tokena [SEP] do rozdzielania zdań. Rozważmy na przykład następujące zdanie:

Pizza with four topping.

Tokenizacja BERT z poprzedniego zdania wygląda następująco:

```
['[CLS]', 'pizza', 'with', 'four', 'topping',
'##s', 'and', 'trim', '##ming', '##s', '.',
'[SEP]']
```

Na listingu wyświetlana jest zawartość pliku bert_special_tokens.py, która ilustruje sposób wyświetlania specjalnych tokenów w BERT

```
import transformers
import numpy as np

# instantiate a BERT tokenizer and model:
print("creating tokenizer...")
tokenizer = transformers.BertTokenizer.from_
pretrained('bert-base-uncased', do_lower_case=True)

print("creating model...")
nlp = transformers.TFBertModel.from_pretrained
('bert-base-uncased')

# hidden layer with embeddings:
text1      = "cell phone"
input_ids1 = np.array(tokenizer.encode(text1))
[None,:]
embedding1 = nlp(input_ids1)

print("input_ids1:")
print(input_ids1)
print()

print("tokenizer.sep_token:   ",tokenizer.sep_
token)
print("tokenizer.sep_token_id:",tokenizer.sep_
token_id)
print("tokenizer.cls_token:   ",tokenizer.cls_
token)
print("tokenizer.cls_token_id:",tokenizer.cls_
token_id)


print("tokenizer.pad_token:   ",tokenizer.pad_
token)
print("tokenizer.pad_token_id:",tokenizer.pad_
token_id)
print("tokenizer.unk_token:   ",tokenizer.unk_
token)
print("tokenizer.unk_token_id:",tokenizer.unk_
token_id)
print()
```

Listing rozpoczyna się od dwóch instrukcji importu, a następnie inicjalizuje zmienną tokenizer jako instancję wstępnie wytrenowanego modelu. Następnie zmienna nlp jest inicjalizowana jako instancja wstępnie wytrenowanego modelu. Kolejna część Listingu inicjuje zmienną text1 jako dwuwyrazowy ciąg, a następnie zmienną input-ids1, która zawiera tokeny dla tych dwóch słów oraz dwa tokeny specjalne. Ostatni blok kodu składa się z zestawu instrukcji print(), które wyświetlają kilka tokenów specjalnych i ich wartości token_id. Uruchom kod z Listingu , a zobaczysz następujący wynik:

```
creating tokenizer...
creating model…
input_ids1:
[[ 101 3526 3042  102]]

tokenizer.sep_token:    [SEP]
tokenizer.sep_token_id: 102
tokenizer.cls_token:    [CLS]
tokenizer.cls_token_id: 101
tokenizer.pad_token:    [PAD]
tokenizer.pad_token_id: 0
tokenizer.unk_token:    [UNK]
tokenizer.unk_token_id: 100
```

TOKENIZERY OPARTE NA BERT

Chociaż modele oparte na BERT wykazują znaczne podobieństwo, istnieją różnice między ich tokenizerami. Poniższe podsekcje przedstawiają różnice między tokenizerem BERT a tokenizerem DistilBERT.

Tokenizator BERT

Listing wyświetla zawartość pliku bert_tokenizer.py, która pokazuje, jak BERT tokenizuje zdania w języku angielskim.

```
from transformers import BertTokenizer

tokenizer = BertTokenizer.from_pretrained
('bert-base-uncased')
sentence = "I love Chicago pizza"
encoding = tokenizer.encode(sentence)

print("-> sentence:",sentence)
print("-> encoding:",encoding)
print("-> tokens: ",tokenizer.convert_ids_to_tokens
(encoding))
print()

sentence = 'My favorite word is syzygy'
encoding = tokenizer.encode(sentence)

print("-> sentence:",sentence)
print("-> encoding:",encoding)
print("-> tokens: ",tokenizer.convert_ids_to_tokens
(encoding))
```

Listing rozpoczyna się od instrukcji import, a następnie inicjalizuje zmienną tokenizer jako instancję wstępnie wytrenowanego modelu. Następnie zmienna sentence jest inicjalizowana jako ciąg tekstowy (sentence), a kodowanie zmiennej jest inicjalizowane wynikiem wywołania metody encode() tokenizera. Następnie wyświetlane są wartości wszystkich trzech zmiennych. W kolejnej części Listingu zmienna sentence jest inicjalizowana jako kolejny ciąg tekstowy, a kodowanie jest ponownie inicjalizowane poprzez ponowne wywołanie metody encode(). Ponownie wyświetlane są wartości trzech zmiennych. Uruchom kod w Listingu, a zobaczysz następujący wynik:

```
-> sentence: I love Chicago pizza
-> encoding: [101, 1045, 2293, 3190, 10733, 102]
```

```
-> tokens:   ['[CLS]', 'i', 'love', 'chicago',
'pizza', '[SEP]']

-> sentence: My favorite word is syzygy
-> encoding: [101, 2026, 5440, 2773, 2003, 25353,
9096, 6292, 102]
-> tokens:   ['[CLS]', 'my', 'favorite', 'word',
'is', 'sy', '##zy', '##gy', '[SEP]']
```

Tokenizer DistilBERT

Każda odmiana BERT ma określony typ danych wejściowych i może używać różnych parametrów. Na przykład DistilBERT używa tych samych tokenów specjalnych co BERT, ale nie używa identyfikatorów token_type_id. Lista zawiera zawartość pliku distilbert_tokenizer.py, która pokazuje, że DistilBERT tokenizuje zdania w języku angielskim w taki sam sposób jak BERT.

```
from transformers import DistilBertTokenizer

tokenizer = DistilBertTokenizer.from_pretrained
('distilbert-base-uncased')
sentence = "I love Chicago pizza"
encoding = tokenizer.encode(sentence)

print("-> sentence:",sentence)
print("-> encoding:",encoding)
print("-> tokens: ",tokenizer.convert_ids_to_tokens
(encoding))
print()

sentence = 'My favorite word is syzygy'
encoding = tokenizer.encode(sentence)

print("-> sentence:",sentence)
print("-> encoding:",encoding)
print("-> tokens: ",tokenizer.convert_ids_to_tokens
(encoding))
```

Listing 3 różni się od Listingu 2 dwoma pierwszymi wierszami, w których BERT jest zastępowany przez DistilBERT, a bert-base-uncased przez distilbert-base-uncased. Uruchom kod z Listingu 3, a zobaczysz następujący wynik, identyczny z wynikiem z Listingu 2:

```
-> sentence: I love Chicago pizza
-> encoding: [101, 1045, 2293, 3190, 10733, 102]
-> tokens:   ['[CLS]', 'i', 'love', 'chicago',
'pizza', '[SEP]']

-> sentence: My favorite word is syzygy
-> encoding: [101, 2026, 5440, 2773, 2003, 25353,
9096, 6292, 102]
-> tokens:   ['[CLS]', 'my', 'favorite', 'word',
'is', 'sy', '##zy', '##gy', '[SEP]']
```

ANALIZA NASTRÓJU ZA POMOCĄ DISTILBERTA

Nauczysz się, jak pracować z tokenizerami i modelem DistilBERT. W tej sekcji zobaczysz, jak przeprowadzić klasyfikację tekstu za pomocą DistilBERTA. Klasa potoku w przykładowym kodzie „abstrahuje" wiele szczegółów, ułatwiając wnioskowanie, ale jest mniej elastyczna niż ręczna konfiguracja modelu i tokenizera. Jeśli chcesz dostroić model na podstawie niestandardowych danych,

będziesz potrzebować bardziej szczegółowej konfiguracji niż ta oferowana przez potok. Ponadto przykładowy kod wykorzystuje domyślną konfigurację analizy sentymentu w DistilBERT. Jeśli masz niestandardowy model, zastąp distilbert-base-uncased ścieżką do swojego modelu. Listing wyświetla zawartość pliku distilbert_sentiment_analysis.py, który pokazuje, jak przeprowadzić analizę sentymentu.

```
from transformers import
pipeline, DistilBertTokenizer,
DistilBertForSequenceClassification

# Load DistilBERT tokenizer and model
tokenizer = DistilBertTokenizer.from_pretrained
('distilbert-base-uncased')


model = DistilBertForSequenceClassification.from_
pretrained('distilbert-base-uncased')

# Create the sentiment analysis pipeline
classifier = pipeline('sentiment-analysis',
model=model, tokenizer=tokenizer)

# Test the classifier
sentences = ["This is an amazing product!",
"I don't really like this movie."]
predictions = classifier(sentences)

for text, prediction in zip(sentences,
predictions):
  label = prediction['label']
  score = prediction['score']
  print(f"'{text}' is classified as {label} with
a score of {score:.4f}")
```

Listing rozpoczyna się od instrukcji importu, a następnie inicjalizuje tokenizer zmiennych i model zmiennych z drugiej i trzeciej zaimportowanej klasy. Następnie klasyfikator zmiennych jest inicjowany poprzez wywołanie funkcji pipeline() i określenie zadania sentiment_analysis, a także odwołanie się do wcześniej zdefiniowanego modelu i tokenizera. Kolejna część Listingu definiuje zmienną „zdania" jako listę ciągów znaków (krótkich zdań), a następnie inicjuje zmienną „predykcje" wynikiem wywołania klasyfikatora ze zmienną „zdania". Ostatnia część Listingu zawiera pętlę, która iteruje po zdaniach i odpowiadających im predykcjach. Podczas każdej iteracji zmienne „etykieta" i „score" są inicjowane, a następnie wyświetlane. Uruchom kod z Listingu , a zobaczysz następujący wynik:

```
'This is an amazing product!' is classified as
LABEL_0 with a score of 0.5325
'I don't really like this movie.' is classified as
LABEL_0 with a score of 0.5307
```

Interesującym aspektem poprzedniego wyniku jest wartość LABEL_0 dla etykiety, co sugeruje, że DistilBERT wyświetla zmienną wewnętrzną. W rzeczywistości DistilBERT raportuje bardzo podobne wyniki dla innych zdań, które są wyraźnie wysoce pozytywne lub wysoce negatywne. Jednak DistilBERT to przede wszystkim wstępnie wytrenowany model reprezentacji języka, który jest trenowany w celu przewidywania zamaskowanych tokenów w zdaniu, a gotowy DistilBERT nie jest specjalnie dostrojony do analizy sentymentu. Jest to raczej reprezentacja ogólnego przeznaczenia, która odzwierciedla semantykę języka angielskiego (lub innych języków, w zależności od konkretnego wariantu modelu).

W przypadku konkretnych zadań, takich jak analiza sentymentu, zazwyczaj bierze się wstępnie wytrenowany model DistilBERT i dostraja go do zbioru danych analizy sentymentu.

KODOWANIE BERT: SEKWENCJA KROKÓW

BERT wykonuje następującą sekwencję kroków, z których wszystkie zostały zilustrowane za pomocą fragmentów kodu w poprzednich sekcjach:

Krok 1: Tokenizacja tekstu.

Krok 2: Przypisz tokeny do ich identyfikatorów.

Krok 3: Dodaj specjalne tokeny [CLS] i [SEP].

Jako prosty przykład, zdanie „Dostałem książkę" ma łącznie sześć tokenów (cztery tokeny słów oraz token początkowy i końcowy), wraz z następującymi indeksami:

[CLS]	**101**
i	1,045
got	2,288
a	1,037
book	2,338
[SEP]	**101**

Na liście wyświetlona jest zawartość pliku bert_encoding_plus.py ilustrująca sposób wyświetlania specjalnych tokenów w BERT.

```
import transformers
import numpy as np
# instantiate a BERT tokenizer and model:
print("creating tokenizer...")
tokenizer = transformers.BertTokenizer.from_
pretrained('bert-base-uncased', do_lower_case=True)
print("creating model...")
nlp = transformers.TFBertModel.from_pretrained
('bert-base-uncased')
```

```
text="When were you last outside? I have been
inside for 2 weeks."
encoding = tokenizer.encode_plus(
   text,
   max_length=32,
   add_special_tokens=True, # Add '[CLS]' and
'[SEP]'
   return_token_type_ids=False,
   pad_to_max_length=True,
   return_attention_mask=True,
   return_tensors='pt',  # Return PyTorch tensors
)

print("encoding.keys():")
print(encoding.keys())
print()

print("len(encoding['input_ids'][0]):")
print(len(encoding['input_ids'][0]))
print()

print("encoding['input_ids'][0]:")
print(encoding['input_ids'])
print()

print("len(encoding['attention_mask'][0]):")
print(len(encoding['attention_mask'][0]))
print()

print("encoding['attention_mask']:")
print(encoding['attention_mask'])
print()

print("tokenizer.convert_ids_to_tokens
(encoding['input_ids'][0]):")
print(tokenizer.convert_ids_to_tokens
(encoding['input_ids'][0]))
print()
```

Listing rozpoczyna się od dwóch instrukcji importu, a następnie inicjuje zmienne tokenizer i nlp w taki sam sposób, jak w poprzednich przykładach kodu. Następnie zmienna text jest inicjowana jako ciąg tekstowy, po którym następuje kodowanie zmiennej, które działa jak konfiguracyjny „przechowacz" parametrów i ich wartości. Ostatnia część Listingu składa się z sześciu par instrukcji print(), z których każda wyświetla parę parametr/wartość zdefiniowaną w zmiennej kodowania. Uruchom kod z Listingu , a zobaczysz następujący wynik:

```
creating tokenizer...
creating model…

encoding.keys():
dict_keys(['input_ids', 'attention_mask'])

len(encoding['input_ids'][0]):
32

encoding['input_ids'][0]:
tensor([[ 101, 2043, 2020, 2017, 2197, 2648,
1029, 1045, 2031, 2042, 2503, 2005, 1016, 3134,
1012,  102,    0,    0,    0,    0,    0,    0,
0,    0,    0,    0,    0,    0,    0,    0,
0,    0]])

len(encoding['attention_mask'][0]):
32

encoding['attention_mask']:
tensor([[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0]])

tokenizer.convert_ids_to_tokens(encoding['input_
ids'][0]):
['[CLS]', 'when', 'were', 'you', 'last', 'outside',
'?', 'i', 'have', 'been', 'inside', 'for', '2',
'weeks', '.', '[SEP]', '[PAD]', '[PAD]', '[PAD]',
'[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]',


'[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]',
'[PAD]', '[PAD]', '[PAD]']
```

PODOBIEŃSTWO ZDAŃ W PROGRAMIE BERT

Word2vec i GloVe wykorzystują osadzenia słów do znajdowania podobieństwa semantycznego między dwoma słowami. Zdania zawierają jednak dodatkowe informacje, a także relacje między wieloma słowami. Znany przykład, który wyraźnie pokazuje potrzebę świadomości kontekstowej, ilustruje następująca para zdań:

• Pies nie przeszedł przez ulicę, ponieważ była zbyt wąska.

• Pies nie przeszedł przez ulicę, ponieważ był zbyt zmęczony.

Jedna z technik określania podobieństwa zdań polega na obliczeniu średniej osadzeń słów w każdym zdaniu, a następnie obliczeniu podobieństwa cosinusowego otrzymanej pary osadzeń słów. Ta metoda jest prosta i wydajna obliczeniowo, ale nie zawsze uchwyca niuanse semantyczne zdań tak skutecznie, jak bardziej zaawansowane metody, ponieważ nie uwzględnia kolejności słów i relacji między nimi. Traktuje ona zdanie jako „worek słów". Jednak pomimo swojej prostoty, w wielu przypadkach może on dawać całkiem dobre rezultaty, zwłaszcza w połączeniu z wysokiej jakości osadzeniem słów. Należy pamiętać, że chociaż wiele technik osadzania słów, w tym Word2Vec i GloVe, jest trenowanych w sposób nienadzorowany z wykorzystaniem dużych korpusów, nie jest z natury konieczne, aby wszystkie techniki osadzania słów były nienadzorowane. Istnieją również nadzorowane i półnadzorowane techniki osadzania słów.

Kontekst słów w BERT

Listing wyświetla zawartość pliku bert_context.py, która ilustruje, jak BERT generuje inny wektor słów dla tego samego słowa używanego w innym kontekście. Aby uruchomić kod z Listingu , upewnij się, że zainstalowano bibliotekę transformers, wpisując następujące polecenie w powłoce poleceń:

pip3 install transformers

Uwaga: Ten kod pobiera model BERT o rozmiarze 536 MB.

```
import transformers

text1 = "cell phone"

# instantiate a BERT tokenizer and model:
tokenizer = transformers.BertTokenizer.from_
pretrained('bert-base-uncased', do_lower_case=True)

nlp = transformers.TFBertModel.from_pretrained
('bert-base-uncased')

# hidden layer with embeddings:
input_ids1 = np.array(tokenizer.encode(text1))
[None,:]
embedding1 = nlp(input_ids1)

# display text1 and its context:
print("text1:",text1)
print("embedding1[0][0]:")
print(embedding1[0][0])
print()

text2 = "cell mate"
# hidden layer with embeddings:
input_ids2 = np.array(tokenizer.encode(text2))
[None,:]
embedding2 = nlp(input_ids2)

# display text2 and its context:
print("text2:",text2)
print("embedding2[0][0]:")
print(embedding2[0][0])
```

Listing rozpoczyna się od instrukcji importu, a następnie inicjuje zmienne tokenizer, nlp, input_ids1 i embedding1 w dokładnie taki sam sposób, jak w poprzednich przykładach kodu. Kolejny blok kodu wyświetla wartości text1 i embedding1[0][0]. Kolejny fragment Listingu jest praktycznie taki sam, jak poprzedni blok kodu, oparty na zamianie tekstu 1 na tekst 2. Wynik Listingu jest następujący:

```
text1: cell phone
embedding1[0][0]:
tf.Tensor(
[[-0.30501425  0.14509355 -0.18064171 ...
-0.3127299  -0.12173399
  -0.09033043]
 [ 0.80547976 -0.15233847  0.61319923 ...
-0.7498784   0.00167803
  -0.11698578]
 [ 1.0339862  -0.66511637 -0.17642722 ...
-0.24407595  0.03978422
  -0.8694502 ]
 [ 0.87851435  0.10932285 -0.27658027 ...
0.18180653 -0.5829581
  -0.34113947]], shape=(4, 768), dtype=float32)

text2: cell mate
embedding2[0][0]:
tf.Tensor(
[[-0.24141303  0.1146469  -0.13710016 ...
-0.2908613  -0.04577148
   0.2965925 ]
 [ 0.05608664 -1.0035615   0.12738925 ...
-0.30271983  0.17530476
   0.7245784 ]
 [ 0.2818157  -0.28047347 -0.6547173  ...
0.04996978  0.01698243
   0.03285426]
 [ 1.039136    0.12364347 -0.2661501  ...
0.09439699 -0.7794917
  -0.24966209]], shape=(4, 768), dtype=float32)
```

Wystawienie listy generuje również następujący komunikat informacyjny:

```
Some weights of the model checkpoint at bert-
base-uncased were not used when initializing
TFBertModel: ['nsp___cls', 'mlm___cls']


 - This IS expected if you are initializing
 TFBertModel from the checkpoint of a model trained
 on another task or with another architecture
 (e.g. initializing a BertForSequenceClassification
 model from a BertForPretraining model).
 - This IS NOT expected if you are initializing
 TFBertModel from the checkpoint of a model that
 you expect to be exactly identical (initializing
 a BertForSequenceClassification model from a
 BertForSequenceClassification model).
 All the weights of TFBertModel were initialized
 from the model checkpoint at bert-base-uncased.
 If your task is similar to the task the model of
 the ckeckpoint was trained on, you can already
 use TFBertModel for predictions without further
 training.
```

Czym jest Sentence-BERT?

Sentence-BERT to wstępnie wytrenowany model LLM oparty na modelu BERT LLM, który wykonuje sieci zdaniowe BERT, modyfikując architektury BERT i RoBERTa. Następnie wyjściem jest osadzenie zdań o stałym rozmiarze, generowane za pomocą następujących technik:

1. Dodanie operacji łączenia do wyjścia transformatorów.

2. Obliczenie średniej wszystkich wektorów wyjściowych.

3. Obliczenie wartości maksymalnej wektorów wyjściowych w czasie.

Dostępnych jest kilka wstępnie wytrenowanych modeli Sentence-BERT

Teraz, gdy zapoznałeś się z przykładem, w którym BERT generuje inny wektor słów dla słowa używanego w innym kontekście, przyjrzyjmy się tokenom BERT, które są tematem następnej sekcji.

GENEROWANIE TOKENÓW BERT (1)

Listing przedstawia zawartość pliku bert_tokens1.py, który ilustruje sposób konwersji ciągu tekstowego na ciąg zgodny z BERT, a następnie tokenizacji tego ostatniego na tokeny BERT.

```
from transformers import BertTokenizer, BertModel

tokenizer = BertTokenizer.from_pretrained
('bert-base-uncased')

text1 = "Pizza with four toppings and trimmings."
marked_text1 = "[CLS] " + text1 + " [SEP]"
tokenized_text1 = tokenizer.tokenize(marked_text1)

print("input sentence #1:")
print(text1)
print()

print("Tokens from input sentence #1:")
print(tokenized_text1)
print()

print("Some tokens in BERT:")
print(list(tokenizer.vocab.keys())[1000:1020])
print()
```

Listing importuje BertTokenizer i BertModel, a pierwszy z nich używa do zainicjowania zmiennej tokenizer. Następnie zmienna text1 jest inicjowana ciągiem tekstowym, a marked_text1 poprzedza [CLS] ciągiem tekstowym text1, a następnie dodaje [SEP] do text1. Ostatnią inicjowaną zmienną jest tokenized_text1, do której przypisywany jest wynik wywołania metody tokenizer() dla zmiennej marked_text1. Kolejne trzy bloki instrukcji print() wyświetlają odpowiednio zawartość tokenów text1, tokenized_text1 i zakres 20 tokenów BERT. Uruchom kod w Listingu, a zobaczysz następujący wynik:

```
input sentence #1:
Pizza with four toppings and trimmings.

Tokens from input sentence #1:
['[CLS]', 'pizza', 'with', 'four', 'topping',
'##s', 'and', 'trim', '##ming', '##s', '.',
'[SEP]']

Some tokens in BERT:
['"', '#', '$', '%', '&', "'", '(', ')', '*', '+',
',', '-', '.', '/', '0', '1', '2', '3', '4', '5']
```

GENEROWANIE TOKENÓW BERT (2)

Listing przedstawia zawartość pliku bert_tokens2.py, który ilustruje sposób konwersji ciągu tekstowego na ciąg zgodny z BERT, a następnie tokenizacji tego ostatniego na tokeny BERT.

```
from transformers import BertTokenizer, BertModel

tokenizer = BertTokenizer.from_pretrained
('bert-base-uncased')

text2 = "I got a book and after I book for an hour,
it's time to book it."
marked_text2 = "[CLS] " + text2 + " [SEP]"
tokenized_text2 = tokenizer.tokenize(marked_text2)

print("input sentence #2:")
print(text2)
print()

print("Tokens from input sentence #2:")
print(tokenized_text2)
print()

# Map token strings to their vocabulary indices:
indexed_tokens2 = tokenizer.convert_tokens_to_ids
(tokenized_text2)

# Display the words with their indices:
for pair in zip(tokenized_text2, indexed_tokens2):
  print('{:<12} {:>6,}'.format(pair[0], pair))
```

Pierwsza połowa Listingu jest niemal identyczna z pierwszą połową Listingu 2, używa bowiem zmiennej text2 zamiast text1.

Następna część Listingu zawiera dwa bloki instrukcji print(), które wyświetlają odpowiednio zawartość text2 i tokenized_text2. Kolejny fragment kodu inicjuje zmienną indexed_tokens2, aby uzyskać wynik konwersji tokenów w tokenized_text2 na wartości id. Ostatnia część Listingu 6.8 zawiera pętlę wyświetlającą tokeny i powiązane z nimi wartości id. Wynik Listingu znajduje się poniżej:

```
input sentence #2:
I got a book and after I book for an hour, it's
time to book it.

Tokens from input sentence #2:
['[CLS]', 'i', 'got', 'a', 'book', 'and', 'after',
'i', 'book', 'for', 'an', 'hour', ',', 'it', "'",
's', 'time', 'to', 'book', 'it', '.', '[SEP]']

[CLS]          101
i            1,045
got          2,288
a            1,037
book         2,338
and          1,998
after        2,044
i            1,045
book         2,338
for          2,005
an           2,019
hour         3,178
,            1,010
it           2,009
'            1,005
s            1,055
time         2,051
to           2,000
book         2,338
it           2,009
.            1,012
[SEP]          102
```

RODZINA BERTÓW

BERT (od Google) dał początek niezwykłemu zestawowi wariantów oryginalnego modelu BERT, z których każdy oferuje interesujące funkcje. Niektóre z tych wariantów są wymienione tutaj:

- ALBERT (Google)
- Transformatory dwukierunkowe i autoregresyjne (BART)
- BERT (Google)
- BIO BERT
- BERT kliniczny
- DeBERTa (Microsoft)
- DistilBERT (Hugging Face)
- DOC BERT
- KeyBERT
- Google SMITH
- SBERT
- TinyBERT

• VisualBERT

• XLM-R

• XLNET

Niektóre modele z powyższej listy omówiono w kolejnych podrozdziałach, a więcej informacji na temat modeli, które nie zostały omówione w tym rozdziale, można znaleźć w wyszukiwarce online. Dodatkowo, RoBERTa jest omówiona w kolejnej sekcji. ALBERT ALBERT (od Google Research i Toyota Technological Institute) to akronim od A Lite BERT for Self-Supervised Learning of Language Representations (Lite BERT for Samodzielne Uczenie się Reprezentacji Języka). ALBERT i RoBERTa są znacznie mniejsze niż BERT i znacznie bardziej wydajne. Głównym celem ALBERT jest przewyższenie dokładności BERT-a w testach porównawczych. Zazwyczaj oznacza to zwiększenie rozmiaru modelu i wprowadzenie modyfikacji w procesie uczenia, aby zapewnić skalowalność: to znaczy, aby umożliwić modelowi zwiększenie dokładności bez nadmiernego dopasowania. Przypomnijmy, że osadzenia słów w BERT-cie to wektory liczb zmiennoprzecinkowych o długości 1x768. Natomiast początkowe osadzenia słów w ALBERT-cie mają tylko 128 cech (1x128 wektorów). Co ciekawe, ALBERT zawiera również macierz, która skaluje osadzenia słów o długości 1x128 do 1x768. Co więcej, ALBERT ma 12 warstw i ukryty rozmiar 4096, co stanowi połowę głębokości i cztery razy szerokość BERT-Large.

Według repozytorium Github dla ALBERT (https://github.com/google-research/ALBERT),

„ALBERT wykorzystuje techniki redukcji parametrów, które umożliwiają konfiguracje na dużą skalę, przezwyciężają dotychczasowe ograniczenia pamięci i zapewniają lepsze zachowanie w odniesieniu do degradacji modelu".

W szczególności ALBERT (w przeciwieństwie do BERT) współdzieli swoje parametry we wszystkich warstwach, co zmniejsza ich liczbę, ale nie ma wpływu na czas treningu i wnioskowania. Ponadto ALBERT wykorzystuje faktoryzację macierzy osadzania, która dodatkowo zmniejsza liczbę parametrów. Należy pamiętać o kilku szczegółach dotyczących techniki współdzielenia parametrów stosowanej przez ALBERT. Po pierwsze, chociaż współdzielone parametry rzeczywiście zmniejszają liczbę unikalnych parametrów, czas wykonania podczas etapu treningu lub etapu wnioskowania (predykcji) jest nadal porównywalny z BERT. Po drugie, chociaż współdzielone parametry skutkują niższą dokładnością, dokładność ta jest nadal całkiem dobra. Po trzecie, redukcja parametrów pozwala na większą skalowalność pod względem głębokości (tj. liczby podkomponentów w komponencie kodera) i szerokości osadzeń. Kolejną cechą wyróżniającą ALBERT jest wykorzystanie SOP, co stanowi ulepszenie w stosunku do NSP stosowanego w BERT. Wreszcie, ALBERT nie wykorzystuje współczynnika dropoutów, co dodatkowo zwiększa pojemność modelu. ALBERT wykorzystuje zarówno maskowanie całych słów, jak i „maskowanie n-gramów", gdzie to drugie odnosi się do maskowania wielu kolejnych słów. Oto fragment kodu dla ALBERT:

```
from transformers import AlbertForMaskedLM,
AlbertTokenizer

model1 = AlbertForMaskedLM.from_pretrained
('albert-xxlarge-v1')
tokenizer = AlbertTokenizer.from_pretrained
('albert-xxlarge-v1')

model2 = AlbertForMaskedLM.from_pretrained
('albert-xxlarge-v2')
tokenizer = AlbertTokenizer.from_pretrained
('albert-xxlarge-v2')
```

ALBERT osiąga wyższe wyniki w testach porównawczych niż BERT, XLNet i RoBERTa, ale tylko w przypadku największego rozmiaru ALBERT-xxlarge, podczas gdy ALBERT-Base generalnie ma porównywalną wydajność z BERT-Base. Jako ciekawostkę, oto daty wydania ALBERT, XLNet i RoBERTa w 2019 roku:

• ALBERT: 26.09.2019 (Google)

• XLNet: 19.06.2019 (Google + CMU)

• RoBERTa: 26.07.2019 (U of W + Facebook)

BART

BART to model składający się z trzech głównych komponentów: kodera (12 warstw), dekodera (12 warstw) i lm_head (warstwy liniowej). BART doskonale nadaje się do generowania tekstu: może rekonstruować tekst zawierający tokeny [MASK], więc jest podobny do autokodera. Co więcej, jego wydajność jest zbliżona do RoBERTa. Dostęp do sekcji modelu BART można również uzyskać, wykonując wywołania programowe, takie jak model.model.encoder, model.model.decoder i model.lm_head. Jednak Hugging Face posiada wbudowane funkcje dostępu do kodera i dekodera, zapewniające dodatkową elastyczność. Dla porównania, BART może również przewyższać GPT-2. BART może być używany do abstrakcyjnego podsumowania tekstu, co jest zadaniem trudniejszym niż podsumowanie ekstraktywne. W rzeczywistości, w pewnym momencie BART był modelem SOTA do podsumowania tekstu. Jeśli nie znasz podsumowania tekstu, znajdziesz je bardziej szczegółowo na stronie https://pub.towardsai.net/a-full-introduction-on-text-summarization-using-deep-learning-with-sample-code-ft-huggingface-d21e0336f50c

BioBERT

BioBERT to model oparty na BERT, który został wstępnie wytrenowany na danych biomedycznych. Można go dostosować do różnych zadań specyficznych dla biomedycyny, takich jak NER i odpowiadanie na pytania (między innymi). Dwa dobrze znane zbiory danych do wstępnego trenowania BioBERT to PubMed i PubMed Central, które zawierają odpowiednio obszerny zbiór cytowań i artykułów biomedycznych. BioBERT można dostosować do wykonywania zadań specyficznych dla biomedycyny, takich jak NER (co dotyczy również ClinicalBERT). BioBERT można pobrać tutaj: https://github.com/naver/biobert-pre-trained.

ClinicalBERT

ClinicalBERT to model oparty na BERT, który został wstępnie wytrenowany na korpusie klinicznym zawierającym szczegółowe informacje dotyczące pacjentów. Na przykład, model zawiera informacje dotyczące objawów pacjenta, diagnozy pacjenta, aktywności i leczenia. Ponieważ dokumenty kliniczne zawierają nomenklaturę („żargon") specyficzną dla branży, ClinicalBERT został wstępnie wytrenowany na zestawie dokumentów klinicznych o nazwie MIMIC-III. Co więcej, ClinicalBERT można precyzyjnie dostroić do różnych zadań, takich jak zadania związane z predykcją.

deBERTa (Przewyższanie ludzkiej dokładności)

Model deBERTa firmy Microsoft faktycznie przewyższył ludzką dokładność. Architektura tego modelu obejmuje 48 warstw transformatorowych z 1,5 miliarda parametrów. Ten model ma wynik GLUE wynoszący 90,8 i wynik SuperGLUE wynoszący 89,9, co przewyższa wynik wydajności człowieka wynoszący 89,8. W razie potrzeby można wyszukać online szczegółowe informacje dotyczące GLUE i SuperGLUE. Microsoft ogłosił również zamiar zintegrowania DeBERTa z modelem reprezentacji języka

naturalnego Turinga Turing NLRv4 (również od Microsoftu). Modele Turinga są wszechobecne w ekosystemie Microsoftu, w tym w produktach takich jak Bing i Azure Cognitive Services.

DistilBERT

DistilBERT to mniejsza odmiana BERT, która zawiera 66 milionów parametrów, co stanowi 40% liczby parametrów BERT-Base (który ma 110 milionów parametrów). DistilBERT wykorzystuje technikę znaną jako destylacja (tj. trenowanie mniejszego modelu w celu aproksymacji większego modelu) do aproksymacji BERT. Mimo to, DistilBERT osiąga ponad 95% dokładności BERT i jest o 60% szybszy niż BERT-Base, co czyni go użytecznym w transferze wiedzy. Najnowsze postępy Neural Magic, wykorzystujące techniki oparte na rozproszonych danych, sprawiły, że BERT-Light jest szybszy niż DistilBERT bez utraty dokładności. Destylacja wiedzy obejmuje mały model (zwany „uczniem"), który jest trenowany w celu naśladowania większego modelu lub zespołu modeli (zwanego „nauczycielem"). DistilBERT jest przykładem sieci destylowanej: zawiera połowę liczby warstw BERT. Dla przykładu, poniżej znajduje się przykład utworzenia tokenizera DistilBERT:

```
import transformers
tokenizer = transformers.AutoTokenizer.from_
pretrained('distilbert-base-uncased',
do_lower_case=True)
```

Oto kolejna technika tworzenia tokenizatora DistilBERT:

```
from transformers import DistilBertTokenizer:
tokenizer = DistilBertTokenizer.from.
pretrained('distilbert-base-uncased')
```

Google SMITH

Model SMITH od Google to model do analizy dokumentów. W bardzo uproszczonym opisie, model SMITH jest trenowany do rozumienia fragmentów w kontekście całego dokumentu. Natomiast BERT jest trenowany do rozumienia słów w kontekście zdań. Jednak model SMITH (który przewyższa BERT) uzupełnia BERT, wykonując główne operacje, które nie są możliwe w BERT.

TinyBERT

TinyBERT opiera się na destylacji i wykorzystuje nową, dwuetapową strukturę uczenia, która przeprowadza destylację transformatorową zarówno podczas wstępnego treningu, jak i na etapach uczenia się zadań. W rezultacie TinyBERT przechwytuje wiedzę ogólną i wiedzę dotyczącą zadań w BERT. Oto kilka szczegółów dotyczących TinyBERT:

- zawiera tylko 4 warstwy
- wydajność jest większa niż 96% wydajności BERT-Base
- 7,5 razy mniejszy niż BERT-Base
- 9,4 razy szybszy niż BERT-Base pod względem wnioskowania

VideoBERT

VideoBERT to oparty na BERT model przetwarzania filmów, który polega na generowaniu tokenów językowych i wizualnych. Tokeny językowe można wyodrębnić za pomocą ASR (automatycznego rozpoznawania mowy), a tokeny wizualne można utworzyć, najpierw pobierając próbki klatek obrazu z filmu, a następnie konwertując je na tokeny wizualne. Następnie tokeny językowe i wizualne są łączone w zdania za pomocą specjalnych tokenów specyficznych dla BERT, takich jak [CLS], [MASK] i

[SEP], po czym przeprowadzany jest algorytm MLM. Jednak zamiast NSP (prognozowanie następnego zdania), VideoBERT przeprowadza sprawdzenie dopasowania, które określa, czy dany ciąg tekstowy i sąsiadujący z nim film mają sens.

VisualBERT

VisualBERT to interesujący model oparty na BERT, który łączy obrazy i tekst. Model ten wykorzystuje uwagę do wykrywania „dopasowania" między obszarami obrazu a tekstem. Istnieje kilka implementacji VisualBERT opartych na społeczności, a ta z największą liczbą gwiazdek na GitHubie znajduje się pod adresem https://github.com/uclanlp/visualbert.

XLNET

XLNet to autoregresyjny, wstępnie wytrenowany model, który przewyższa BERT w wielu zadaniach. XLNet wykorzystuje permutacyjne modelowanie języka (PLM), aby pokonać ograniczenia zadania MLM wykonywanego przez BERT. W przeciwieństwie do BERT, XLNet nie ma żadnych ograniczeń wejściowych. XLNet zawiera również maskę uwagi, która umożliwia XLNet zrozumienie kolejności wyrazów, a także dwustrumieniową samouważność. Ujmując rzecz bardziej ogólnie, biorąc pod uwagę sekwencję wejściową S składającą się z N tokenów, XLNet oblicza prawdopodobieństwo, że i-ty token w S wystąpi w pierwszych i tokenach S, co jest oznaczane jako $P(x_i|x<i)$. To prawdopodobieństwo jest obliczane dla wszystkich możliwych permutacji tokenów w sekwencji S. Przypomnijmy, że istnieje N! możliwych permutacji sekwencji N tokenów.

Uwaga: Dla dowolnego zdania z N tokenami XLNet przetwarza N! zdań, które są permutacjami danego zdania.

Wady XLNet

Pierwszą wadą XLNet jest jego złożoność obliczeniowa, która wymaga dłuższego czasu uczenia w porównaniu z BERT. Po drugie, XLNet może działać słabiej w przypadku krótszych zdań wejściowych. Etap wstępnego uczenia XLNet koncentruje się na długich sekwencjach wejściowych, a permutacyjne modelowanie języka ma na celu wychwycenie długoterminowych zależności. Po trzecie, chociaż XLNet wykorzystuje informacje pozycyjne, odbywa się to w inny sposób niż w tradycyjnych podejściach. W podejściu PLM model uczy się przewidywać tokeny w różnej kolejności, a nie tylko w oryginalnej kolejności w zdaniu, co może potencjalnie prowadzić do pewnego rodzaju „rozbieżności" w sposobie wykorzystania informacji pozycyjnych. Nadal jednak wykorzystuje kodowanie pozycyjne, aby zachować pewien sens informacji pozycyjnych, i jest zaprojektowany do nauki kontekstu dwukierunkowego, co może pomóc mu zrozumieć relacje między różnymi pozycjami w sekwencji. Jak wybrać model oparty na BERT. Istnieje wiele kryteriów, które należy wziąć pod uwagę przy podejmowaniu decyzji o tym, który model zastosować w swoim środowisku. Niektóre z nich wymieniono poniżej:

• dostępność sprzętu i koszt

• uczenie „od zera" a dostrajanie

• rodzaj zadań w dół

• dostępność modeli publicznych

BERT to dobry wybór, jeśli nie masz wysokich wymagań priorytetowych. Jeśli jednak potrzebujesz szybszej prędkości wnioskowania i możesz jednocześnie poświęcić pewną dokładność metryk predykcyjnych, rozważ DistilBERT jako punkt wyjścia. Z kolei, jeśli potrzebujesz najlepszych metryk

predykcyjnych, lepszym wyborem będzie RoBERTA (od Facebooka). Kolejną opcją jest XLNet, który może zapewnić lepsze wyniki w dłuższej perspektywie.

WSPÓŁPRACA Z ROBERTĄ

RoBERTA (od Facebooka) to akronim od A Robustly Optimized BERT Pre-training Approach (Roberta – stabilnie zoptymalizowane podejście do wstępnego trenowania BERT). Architekci RoBERTa wprowadzili następujące ulepszenia do BERT, aby stworzyć RoBERTa:

• zrezygnowali z NSP (prognozowanie następnego zdania)

• wykorzystują „wariant" GPT-2 BPE (nie oryginalny BPE)

• zastosowali dynamiczne maskowanie zamiast statycznego

• przeprowadzono trening na dłuższych sekwencjach.

• wykorzystano 160 GB danych treningowych zamiast 16 GB (BERT)

• próbki tekstu wykorzystują pełne zdania (a nie rozpiętości tekstu)

• nakładanie maski na losowo wybrane zestawy tokenów podczas każdej iteracji

• inny schemat tokenizacji

RoBERTa wykorzystuje strategię maskowania języka BERT, wraz z pewnymi modyfikacjami niektórych hiperparametrów BERT. Należy zauważyć, że RoBERTa został wytrenowany na korpusie, który jest co najmniej 10 razy większy niż korpus BERT. W przeciwieństwie do BERT, RoBERTa nie wykonuje zadania NSP, co faktycznie usprawnia procedurę treningu. Zamiast tego RoBERTa wykorzystuje technikę zwaną maskowaniem dynamicznym, w której różne tokeny są maskowane w różnych iteracjach treningowych podczas procesu treningu. W rezultacie RoBERTa przewyższa zarówno BERT, jak i XLNet w wynikach testów porównawczych GLUE.

TŁUMACZENIE NA JĘZYK WŁOSKI I JAPOŃSKI

Ta sekcja zawiera przykład kodu, który tłumaczy zdanie z języka angielskiego na włoski i japoński. Na liście znajduje się zawartość pliku bert_translate.py ilustrująca sposób przetłumaczenia ciągu tekstowego z języka angielskiego na włoski.

```
from transformers import AutoTokenizer,
AutoModelForSeq2SeqLM

# English to Italian:
tokenizer = AutoTokenizer.from_pretrained
("Helsinki-NLP/opus-mt-en-it")
model = AutoModelForSeq2SeqLM.from_
pretrained("Helsinki-NLP/opus-mt-en-it")
text = "I love deep dish pizza!"
tokenized_text = tokenizer.prepare_seq2seq_batch
([text], return_tensors='pt')

# Perform translation and decode the output
translation = model.generate(**tokenized_text)
```

```
translated_text = tokenizer.batch_decode
(translation, skip_special_tokens=True)[0]

# Print translation from English to Italian:
print("Initial English text:", text)
print("Italian translation: ", translated_text)

# English to Japanese:
tokenizer = AutoTokenizer.from_pretrained
("Helsinki-NLP/opus-mt-en-jap")
model = AutoModelForSeq2SeqLM.from_pretrained
("Helsinki-NLP/opus-mt-en-jap")
text = "I love deep dish pizza!"
tokenized_text = tokenizer.prepare_seq2seq_batch
([text], return_tensors='pt')

# Perform translation and decode the output
translation = model.generate(**tokenized_text)
translated_text = tokenizer.batch_decode
(translation, skip_special_tokens=True)[0]

# Print translation from English to Japanese:
print("Japanese translation:", translated_text)
```

Listing rozpoczyna się od instrukcji import, a następnie inicjuje zmienne tokenizer i model odpowiednio do instancji tokenizera i modelu. Następnie zmienna text jest inicjowana jako ciąg testowy, a zmienna tokenized_text jest inicjowana wynikiem tokenizacji zawartości zmiennej text. Kolejna część Listingu wykonuje tłumaczenie na język włoski, po czym wyświetlany jest przetłumaczony tekst. Kolejny blok tekstu wykonuje podobną operację, tłumacząc początkowy tekst angielski na japoński, a następnie wyświetlany jest wynik. Uruchom kod w Listingu, a zobaczysz następujący wynik:

```
Initial English text: I love deep dish pizza!
Italian translation:  Adoro la pizza dei piatti
profondi!
Japanese translation: わたし は 深 い 偽り の 板 を いだ き,
```

Teraz, gdy zapoznałeś się z przykładem tłumaczenia tekstu między dwiema językami, przyjrzyjmy się modelom LLM, które zapewniają obsługę wielojęzyczną, co jest tematem następnej sekcji.

WIELOJĘZYCZNE MODELE JĘZYKOWE

Obsługa wielojęzyczna w modelach LLM zyskuje na znaczeniu, co czasami może wiązać się z dodatkowymi wyzwaniami. Na przykład język angielski ma ogromny zestaw dostępnych tokenów, podczas gdy wiele języków charakteryzuje się niską dostępnością danych. Jak możemy wytrenować model językowy z ograniczoną liczbą tokenów? Kilka technik zostanie omówionych w następnej sekcji.

Trenowanie wielojęzycznych modeli językowych

Jedna z technik trenowania wielojęzycznych modeli LLM opiera się na koncepcji zwanej transferem międzyjęzykowym.Modele wielojęzyczne obejmują wstępne trenowanie, a następnie XLM-R i transfer międzyjęzykowy. Poniższe repozytorium Github pokazuje, jak korzystać z transferu międzyjęzykowego z języka angielskiego na portugalski: https://github.com/lersouza/cross-lingual-transfer. Modele, których parametry zostały wytrenowane w różnych językach, mogą uczyć się międzyjęzykowych reprezentacji tekstu. Interesującą konsekwencją jest to, że takie reprezentacje umożliwiają wielojęzycznym, wstępnie wytrenowanym modelom wykorzystanie danych nadzorowanych z jednego języka w różnych językach o niskiej ilości danych. Inne podejście obejmuje istniejące wielojęzyczne

modele językowe, co jest tematem następnej sekcji. Wielojęzyczne modele językowe oparte na BERT Istnieją modele oparte na BERT, które obsługują wiele języków, a niektóre z nich wymieniono tutaj:

• m-BERT

• XLM i XLM-R

• XLSR

• mSLAM

• MUSE

• OSCAR

Aby uzyskać więcej informacji na temat programów LLM wymienionych powyżej, wyszukaj je w Internecie. Istnieją dwa typy wstępnie wytrenowanych modeli M-BERT: BERT-Base, wielojęzyczny z uwzględnieniem wielkości liter, oraz BERT-Base, wielojęzyczny bez wielkości liter. Listing 6.10 przedstawia zawartość pliku multi_bert.py, który ilustruje sposób konwersji ciągu tekstowego na ciąg zgodny z BERT, a następnie tokenizacji tego ostatniego na tokeny BERT.

```
from transformers import BertTokenizer, BertModel

model = BertModel.from_pretrained
('bert-base-multilingual-cased')
tokenizer = BertTokenizer.from_pretrained
('bert-base-multilingual-cased')

# Japanese sentence:
sentence = "日本語 が できます か"
print("sentence:")
print(sentence)
print()

# get the sentence tokens:
inputs = tokenizer(sentence, return_tensors="pt")
print("inputs:")
print(inputs)
print()

# pass tokens to the model and get the
representation:
hidden_rep, cls_head = model(**inputs)
print("hidden_rep:")
print(hidden_rep)
print()

print("cls_head:")
print(cls_head)
```

Listing importuje BertTokenizer i BertModel, inicjuje zmienną model jako instancję bert-base-multilingual-cased, a następnie inicjuje zmienną tokenizer. Kolejne trzy bloki instrukcji print() wyświetlają zawartość text1, tokenized_text1 i zakres 20 tokenów BERT. Uruchom kod z Listingu 6.10, a zobaczysz tokeny BERT.

PORÓWNANIE MODELI OPARTYCH NA BERT

Istnieje wiele wariantów oryginalnego modelu BERT, a także modele, które poprawiają możliwości predykcyjne lub wydajność BERT. ALBERT, RoBERTa i DistilBERT należą do najpopularniejszych modeli opartych na BERT. ALBERT i RoBERTa przewyższają BERT pod względem mniejszego rozmiaru i większej szybkości. Z kolei DistilBERT ma mniejszy rozmiar w porównaniu z BERT, a jego dokładność jest niższa zaledwie o 3%. Kilka innych modeli osiąga wydajność SOTA, w tym XLNet, który jest autoregresyjnym modelem językowym rozszerzającym model Transformer-XL. XLNet osiągnął lepszą predykcję niż BERT, wykorzystując permutacyjne modelowanie języka, aby uzyskać wyniki SOTA porównywalne z RoBERTa. Ta technika modelowania osiąga 100% dokładność dla tokenów w losowej kolejności. Przypomnijmy, że BERT przewiduje zamaskowane tokeny, które stanowią 15% wszystkich tokenów. XLNet został wytrenowany z ponad 130 GB danych tekstowych i 512 chipami TPU działającymi przez 2,5 dnia, z których oba są znacznie większe niż BERT. Innym modelem jest Mobile-BERT, który jest podobny do DistilBERT. W porównaniu z BERT-Base, Mobile-BERT jest ponad 4 razy mniejszy i ponad 5 razy szybszy, ale nadal ma porównywalną/podobną wydajność. Trzecim modelem jest BART, który jest wstępnie wytrenowanym modelem, którego wydajność jest porównywalna z RoBERTa w zadaniach NLU. RoBERTa również wprowadziła znaczące ulepszenia. Przypomnijmy, że BERT korzysta z protokołu NSP, podczas gdy RoBERTa wykorzystuje technikę zwaną dynamicznym maskowaniem, w której różne tokeny są maskowane w różnych iteracjach treningowych w trakcie procesu uczenia. Kolejną różnicą w porównaniu z RoBERTa są większe rozmiary pakietów w fazie treningu. Rezultat jest imponujący: RoBERTa przewyższa zarówno BERT, jak i XLNet w wielu zadaniach i zbiorach danych. Inne podejście działa „wbrew" innym modelom językowym: celem jest zmniejszenie rozmiaru modeli językowych opartych na transformatorach przy jednoczesnym osiągnięciu znaczących rezultatów. W szczególności DistilBERT wykorzystuje połowę liczby warstw BERT i unika osadzania w stylu tokenów, a mimo to osiąga poziom wydajności stanowiący 95% wydajności BERT. Ponadto DistilBERT wykorzystuje technikę zwaną destylacją

Jak wytrenować model BERT na laptopie

Jeśli czujesz się zmotywowany i dysponujesz wolnym miejscem na dysku, możesz wytrenować BERT na swoim laptopie. Przejdź do poniższego łącza, gdzie znajdziesz przykład kodu w języku Python zawierający niezbędne szczegóły: https://www.kdnuggets.com/2021/08/train-bert-model-scratch.html Jeśli nie masz wystarczającej ilości miejsca na dysku, po uruchomieniu kodu Pythona z powyższego łącza zobaczysz poniższy komunikat o błędzie (lub coś podobnego), który wskazuje, że potrzebujesz prawie 95 GB miejsca na dysku, aby pobrać ten LLM:

```
f"Not enough disk space. Needed: {size_str(self.
info.size_in_bytes or 0)} (download: {size_
str(self.info.download_size or 0)}, generated:
{size_str(self.info.dataset_size or 0)}, post-
processed: {size_str(self.info.post_processing_size
or 0)})"
OSError: Not enough disk space. Needed: 94.79 GiB
(download: 26.01 GiB, generated: 68.77 GiB,
post-processed: Unknown size)
```

NARZĘDZIA INTERNETOWE DLA BERT

Istnieje kilka bardzo dobrych narzędzi online do eksperymentowania z modelami opartymi na BERT, takich jak exBERT i bertvix, które zostaną pokrótce omówione.

exBERT

Oto formalny opis narzędzia exBERT, zgodnie z opisem na jego stronie głównej: exBERT to narzędzie, które pomaga ludziom prowadzić elastyczne, interaktywne badania i formułować hipotezy dla procesu

wnioskowania wewnętrznego modelu, wspierając analizę szerokiej gamy modeli Hugging Face Transformer. exBERT dostarcza wglądu w znaczenie reprezentacji kontekstowych i uwagi poprzez dopasowanie danych wejściowych określonych przez człowieka do podobnych kontekstów w dużych, adnotowanych zbiorach danych.

BertViz

BertViz to narzędzie do wizualizacji mechanizmu uwagi w modelach opartych na transformatorach, a repozytorium BertViz w serwisie Github znajduje się pod adresem https://github.com/jessevig/bertviz.

CNNViz

Chociaż CNNViz to narzędzie do wizualizacji splotowych sieci neuronowych zamiast modeli powiązanych z BERT, wciąż jest dość interesujące, zwłaszcza jeśli nigdy nie miałeś takiego narzędzia do wizualizacji sieci CNN.

MODELOWANIE TEMATÓW Z UŻYCIEM BERT

Modelowanie tematów polega na znalezieniu zbioru tematów w jednym lub kilku dokumentach. Obejmuje ono znalezienie liczby tematów oraz długości każdego tematu w dokumentach. Modelowanie tematów zawsze było ważne. Przed opracowaniem BERT stosowano różne algorytmy uczenia maszynowego, z których kilka wymieniono poniżej:

• LDA (ukryta alokacja Dirichleta)

• LSA (ukryta analiza semantyczna)

• NMF (nieujemna faktoryzacja macierzy)

Dla porównania, algorytmy klasteryzacji przypisują każde słowo do pojedynczego klastra (tj. relacji jeden do jednego). Modelowanie tematów dopuszcza jednak możliwość przynależności słowa do wielu tematów, co oznacza, że istnieje relacja jeden do wielu między słowem a tematami. W związku z tym modelowanie tematów polega na obliczeniu zbioru prawdopodobieństw przynależności słowa do zbioru tematów. Na szczęście dostępność BERT i powiązanych modeli oznacza, że mamy do dyspozycji wstępnie wytrenowane modele językowe oparte na transformatorach, bez konieczności angażowania kosztownego i czasochłonnego procesu trenowania takich modeli. W następnej sekcji krótko omówiono T5, kolejny zaawansowany model przetwarzania języka naturalnego stworzony przez Google.

CZYM JEST T5?

Transformator transferu tekstu na tekst (T5) to model kodera-dekodera, który konwertuje wszystkie zadania przetwarzania języka naturalnego na format tekstowy. Jego kod do pobrania jest dostępny online:

https://github.com/google-research/text-to-text-transfer-transformer

Możesz również zainstalować T5, wywołując następujące polecenie:

pip install t5[gcp]

T5 jest wstępnie wytrenowany na wielozadaniowej mieszance zadań nadzorowanych i nienadzorowanych i dobrze sprawdza się w zadaniach takich jak tłumaczenie. W 2022 roku T5 był najpopularniejszym pobieraniem z Hugging Face: ponad milion pobrań miesięcznie. T5 jest trenowany z wykorzystaniem techniki zwanej „wymuszaniem nauczyciela”, co oznacza, że do treningu zawsze

wymagana jest sekwencja wejściowa i sekwencja docelowa. Sekwencja wejściowa jest oznaczana za pomocą input_ids, natomiast sekwencja docelowa jest oznaczana za pomocą output_ids, a następnie przekazywana do dekodera. Ponieważ wszystkie zadania (takie jak klasyfikacja, odpowiadanie na pytania i tłumaczenie) wykorzystują ten mechanizm wejścia/wyjścia, ten sam model może być używany do wielu zadań. Ponadto T5 udostępnia kilka przydatnych klas do pracy z modelami T5, takich jak klasa transformers.T5Config, która umożliwia określenie informacji konfiguracyjnych i której wartości domyślne są podobne do architektury T5-small. Inną przydatną klasą jest transformers.T5Tokenizer, która umożliwia skonstruowanie tokenizera T5. T5 różni się od BERT w dwóch istotnych aspektach:

• framework tekst-tekst

• uczenie nadzorowane

• koder-dekoder

• użycie dekodera przyczynowego

• wstępne wytrenowanie za pomocą autokodera odszumiającego

• dane wejściowe mają reprezentację tekst-tekst

• ujednolicone podejście do precyzyjnego dostrajania

• tokenizacja SentencePiece

Chociaż można pobrać przykłady kodu dla T5, początkowo prostszym rozwiązaniem może być eksperymentowanie z T5 w notatniku Google Colaboratory, który można znaleźć w tym artykule i powiązanym z nim repozytorium Github:

https://pedrormarques.wordpress.com/2021/10/21/fine-tuning-a-t5-text-classification-model-on-colab

https://github.com/pedro-r-marques/tutorial-t5-fine-tune

W czerwcu 2022 roku Google udostępniło wszystkie modele Switch Transformer w T5X/JAX jako open source: https://github.com/google-research/t5x. Należy pamiętać, że modele open source obejmują model Switch-C o 1,6 T parametrach oraz model Switch-XXL o 395 B parametrach. T5X to ulepszona implementacja bazy kodu T5 (opartej na Mesh TensorFlow) w JAX i Flax. T5X to modułowe, komponowalne, przyjazne dla badań środowisko do wydajnego, konfigurowalnego, samoobsługowego szkolenia, ewaluacji i wnioskowania modeli sekwencji (zaczynając od języka) w wielu skalach.

PRACA Z PALM

PaLM to oparty na 540 miliardach parametrów model firmy Google, który może uczyć się rozumienia języka, a także wykonywać wiele zadań opartych na NLP. PaLM obsługuje dodatkowe zadania oparte na NLP i może być również trenowany za pomocą licznych akceleratorów sprzętowych. Poniższy link zawiera więcej szczegółów na temat PaLM i innych osiągnięć (takich jak układ H100 firmy Nvidia), a także informacje o LLM (np. Chinchilla) .Co więcej, PaLM wspiera podpowiadanie łańcucha myślowego do wykonywania zadań wieloetapowych. Niewątpliwie PaLM zostanie zastąpiony przez kolejny LLM, który oferuje kombinację wyższej wydajności, mniejszej liczby parametrów lub niższych kosztów całkowitych. Rzeczywiście, dziedzina przetwarzania języka naturalnego (NLP) dynamicznie się zmienia.

Czym jest Pathways?

Pathways to architektura Google, która wykorzystuje inne podejście do trenowania modeli uczenia maszynowego. Z wyjątkiem GPT-3 i innych podobnych systemów, obecne modele sztucznej inteligencji są często trenowane pod kątem pojedynczego zadania. Architektura Pathways umożliwia trenowanym modelom wykonywanie tysięcy zadań. W istocie modele trenowane w architekturze Pathways do wykonywania wielu niepowiązanych zadań będą miały możliwość łączenia tych zadań w celu wykonywania nowych zadań, do których nigdy nie były trenowane.

STRESZCZENIE

Sekcja rozpoczyna się od przykładowego kodu w Pythonie dla specjalnych tokenów BERT. Następnie dowiedziałeś się o tokenizatorach BERT i o tym, jak przeprowadzać analizę sentymentu za pomocą BERT. Następnie poznałeś sekwencję kroków kodowania BERT. Poznałeś również podobieństwo zdań w BERT i sposób generowania tokenów BERT. Ponadto, poznałeś liczne modele LLM należące do rodziny BERT, takie jak ALBERT, BART i DistilBERT. Na koniec, poznałeś narzędzia internetowe dla BERT, modelowanie tematów za pomocą BERT oraz Google PaLM.