O'REILLY®

Helion

Przetwarzanie języka naturalnego z wykorzystaniem transformerów

Budowanie aplikacji językowych za pomocą bibliotek Hugging Face

Lewis Tunstall
Leandro von Werra
Thomas Wolf

Tytuł oryginału: Natural Language Processing with Transformers, Revised Edition

Tłumaczenie: Grzegorz Werner

ISBN: 978-83-289-0711-9

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

Helion S.A.
ul. Kościuszki 1c, 44-100 Gliwice
tel. 32 230 98 63
e-mail: *helion@helion.pl*
WWW: *https://helion.pl* (księgarnia internetowa, katalog książek)

Printed in Poland.

Spis treści

ROZDZIAŁ 1.

Witajcie, transformery

W 2017 r. badacze z firmy Google opublikowali artykuł proponujący innowacyjną architekturę sieci neuronowej do modelowania sekwencyjnego[1]. Architektura ta, nazwana **transformerem**, przewyższała rekurencyjne sieci neuronowe (ang. *recurrent neural network*, RNN) w zadaniach tłumaczenia maszynowego zarówno pod względem jakości tłumaczenia, jak i kosztów treningu.

Jednocześnie efektywna metoda uczenia transferowego znana jako ULMFiT pokazała, że trenowanie sieci z długą pamięcią krótkotrwałą (ang. *long short-term memory*, LSTM) na bardzo dużym i zróżnicowanym korpusie pozwala tworzyć skuteczne klasyfikatory tekstu bez dużej ilości danych opatrzonych etykietami[2].

Postępy te przyczyniły się do powstania dwóch najlepiej znanych transformerów: Generative Pretrained Transformer (GPT)[3] oraz Bidirectional Encoder Representations from Transformers (BERT)[4]. Łącząc architekturę transformera z uczeniem nienadzorowanym, modele te wyeliminowały potrzebę uczenia od podstaw architektur dostosowanych do konkretnych zadań i uzyskały wyniki znacznie przekraczające to, co wcześniej było standardem w NLP. Od czasu wydania GPT i BERT-a pojawiło się wiele innych modeli transformacyjnych; oś czasu z datami wydania najważniejszych transformerów pokazano na rysunku 1.1.

Nie wybiegajmy jednak zanadto do przodu. Aby zrozumieć, co jest nowatorskiego w transformerach, najpierw musimy wyjaśnić:

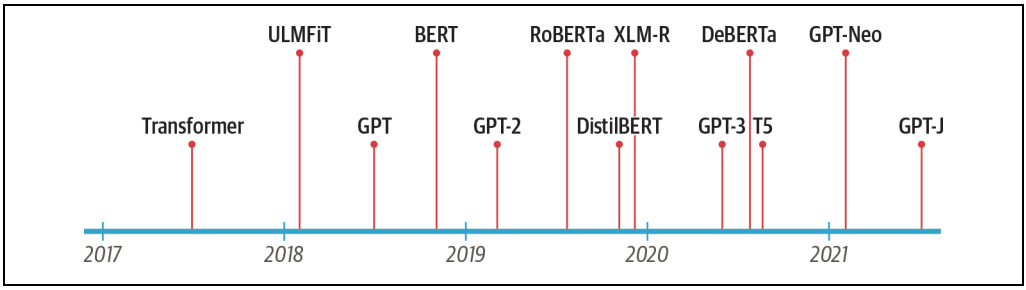
- model kodera-dekodera,
- mechanizmy atencji,
- uczenie transferowe.

[1] A. Vaswani et al., *Attention Is All You Need* (*https://arxiv.org/abs/1706.03762*), 2017. Tytuł ten był tak chwytliwy, że ponad sto późniejszych artykułów (*https://oreil.ly/wT8Ih*) zawierało frazę „all you need" w swoich tytułach!

[2] J. Howard, S. Ruder, *Universal Language Model Fine-tuning for Text Classification* (*https://arxiv.org/abs/1801.06146*), 2018.

[3] A. Radford et al., *Improving language understanding with unsupervised learning* (*https://openai.com/blog/language-unsupervised*), 2018.

[4] J. Devlin et al., *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding* (*https://arxiv.org/abs/1810.04805*), 2018.



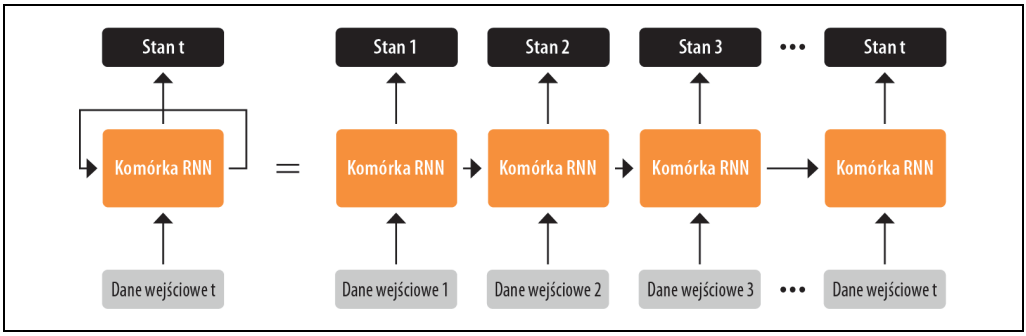
Rysunek 1.1. Oś czasu transformerów

W tym rozdziale wprowadzimy podstawowe koncepcje związane z transformerami, przyjrzymy się niektórym zadaniom, w których uzyskują one szczególnie dobre wyniki, i zakończymy krótkim omówieniem ekosystemu narzędzi oraz bibliotek Hugging Face.

Zacznijmy od zbadania modelu koder-dekoder oraz architektur poprzedzających transformery.

Model koder-dekoder

Przed transformerami najnowszym krzykiem techniki w NLP były architektury rekurencyjne, takie jak sieci LSTM. Architektury te zawierają w połączeniach sieci pętlę zwrotną, która umożliwia propagację informacji między kolejnymi krokami przetwarzania, dzięki czemu nadają się idealnie do modelowania danych sekwencyjnych, takich jak tekst. Jak zilustrowano po lewej stronie rysunku 1.2, model RNN otrzymuje jakieś dane wejściowe (które mogą być słowem lub znakiem), przepuszcza je przez sieć i generuje na wyjściu wektor zwany **stanem ukrytym**. Jednocześnie za pośrednictwem pętli zwrotnej model przekazuje z powrotem do siebie samego pewne informacje, które może wykorzystać w następnym kroku. Widać to wyraźniej, kiedy „rozwinie" się pętlę, jak pokazano po prawej stronie rysunku 1.2: RNN przekazuje informacje o swoim stanie na każdym etapie do następnej operacji w sekwencji. Dzięki temu RNN może śledzić informacje z poprzednich etapów i używać ich do przewidywania wyniku.

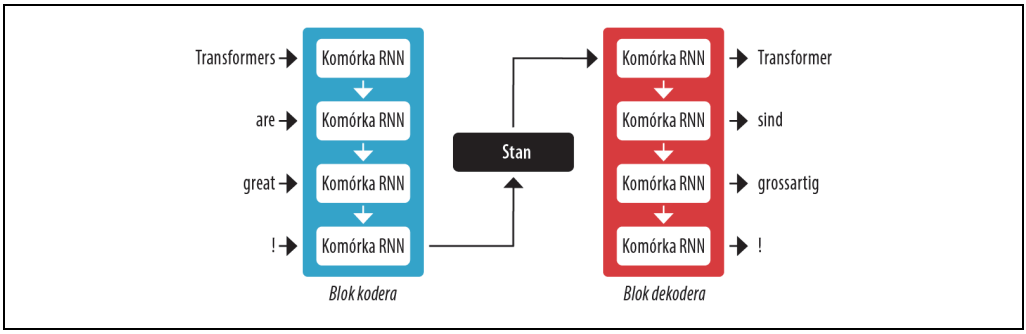


Rysunek 1.2. Pętla RNN rozwinięta w czasie

Architektury te były (i nadal są) szeroko używane do zadań NLP, przetwarzania mowy i analizowania szeregów czasowych. Doskonały opis ich możliwości można znaleźć w poście na blogu Andreja Karpathy'ego, *The Unreasonable Effectiveness of Recurrent Neural Networks* (*https://oreil.ly/Q55o0*).

Jednym z obszarów, w których sieci RNN odegrały ważną rolę, był rozwój systemów tłumaczenia maszynowego, w których celem jest odwzorowanie sekwencji słów w jednym języku na inny język. Zadania tego rodzaju zwykle realizuje się za pomocą architektury **koder-dekoder** lub **sekwencja-sekwencja**[5], dobrze sprawdzającej się w sytuacjach, w których zarówno dane wejściowe, jak i wyjściowe są sekwencjami o dowolnej długości. Zadaniem kodera jest zakodowanie informacji z sekwencji wejściowej w reprezentacji liczbowej, którą często nazywa się **ostatnim stanem ukrytym**. Stan ten następnie przekazuje się do dekodera, który generuje sekwencję wyjściową.

Ogólnie rzecz biorąc, komponenty kodera i dekodera mogą być dowolnymi architekturami sieci neuronowej, które potrafią modelować sekwencję. Na rysunku 1.3 zilustrowano to na przykładzie pary sieci RNN; angielskie zdanie „Transformers are great!" zostaje zakodowane w wektorze stanu ukrytego, który jest następnie dekodowany w celu wygenerowania niemieckiego zdania „Transformer sind grossartig!". Słowa wejściowe przepuszcza się sekwencyjnie przez koder, a słowa wyjściowe są generowane pojedynczo, od góry do dołu.



Rysunek 1.3. Architektura koder-dekoder z parą sieci RNN (ogólnie rzecz biorąc, wykorzystuje się znacznie więcej warstw rekurencyjnych, niż pokazano na tym rysunku)

Choć architektura ta cechuje się elegancką prostotą, jej słabym punktem jest to, że końcowy stan ukryty kodera stwarza **wąskie gardło informacyjne** — musi reprezentować znaczenie całej sekwencji wejściowej, ponieważ jest wszystkim, do czego ma dostęp dekoder podczas generowania danych wyjściowych. Jest to szczególnie problematyczne w przypadku długich sekwencji, ponieważ informacje z początku sekwencji mogą zostać utracone w procesie kompresowania danych w jedną niezmienną reprezentację.

Na szczęście można ominąć to wąskie gardło, zapewniając dekoderowi dostęp do wszystkich stanów ukrytych kodera. Ogólny mechanizm, który to umożliwia, jest nazywany **uwagą** lub **atencją**[6]

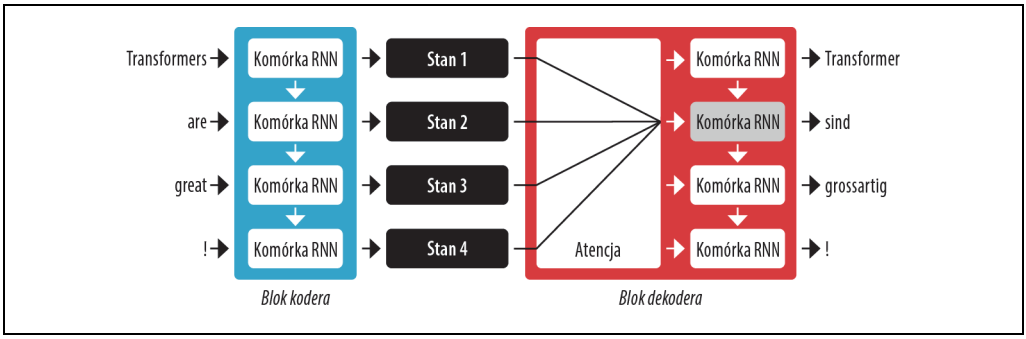
[5] I. Sutskever, O. Vinyals, Q.V. Le, *Sequence to Sequence Learning with Neural Networks* (*https://arxiv.org/abs/1409.3215*), 2014.

[6] D. Bahdanau, K. Cho, Y. Bengio, *Neural Machine Translation by Jointly Learning to Align and Translate* (*https://arxiv.org/abs/1409.0473*), 2014.

i stanowi kluczowy komponent wielu nowoczesnych architektur sieci neuronowych. Kiedy zapoznasz się z mechanizmami atencji opracowanymi z myślą o sieciach RNN, będzie Ci łatwiej zrozumieć jeden z głównych komponentów architektury transformera. Przyjrzymy im się bliżej.

Mechanizmy atencji

Głównym pomysłem stojącym za atencją jest to, że zamiast generować jeden stan ukryty dla sekwencji wejściowej, koder na każdym etapie generuje stan wyjściowy, do którego dostęp ma dekoder. Jednak jednoczesne korzystanie ze wszystkich stanów oznaczałoby, że dekoder otrzymywałby ogromną ilość danych wejściowych, więc potrzebny jest jakiś mechanizm, który określałby priorytety stanów i wybierał te, których należy użyć. Tutaj na scenę wkracza atencja: pozwala ona dekoderowi przypisać różne wagi lub stopnie „uwagi" każdemu ze stanów kodera na każdym etapie dekodowania. Proces ten zilustrowano na rysunku 1.4, który pokazuje wpływ atencji na przewidywanie drugiego tokena w sekwencji wyjściowej.

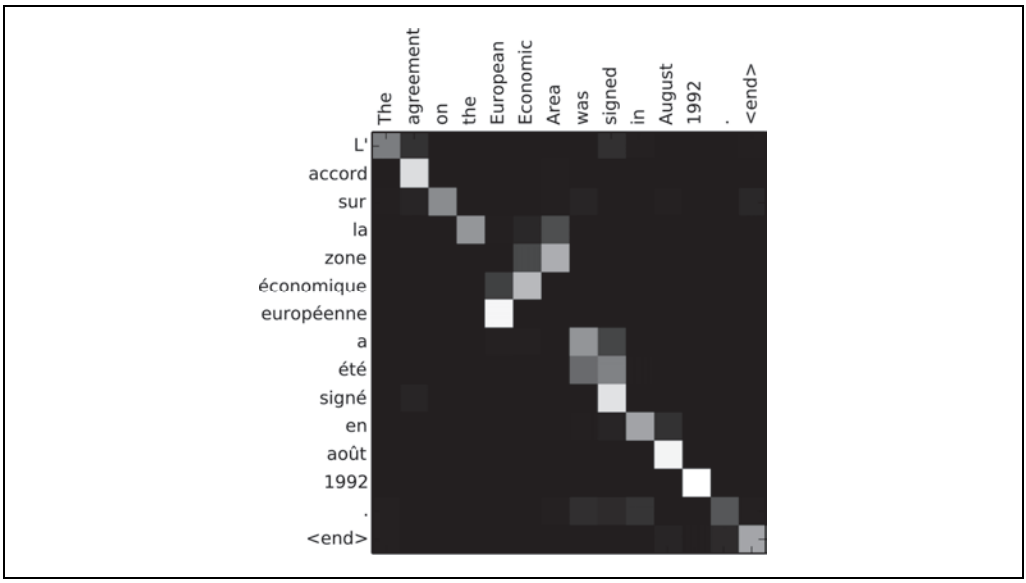


Rysunek 1.4. Architektura koder-dekoder z mechanizmem atencji dla pary sieci RNN

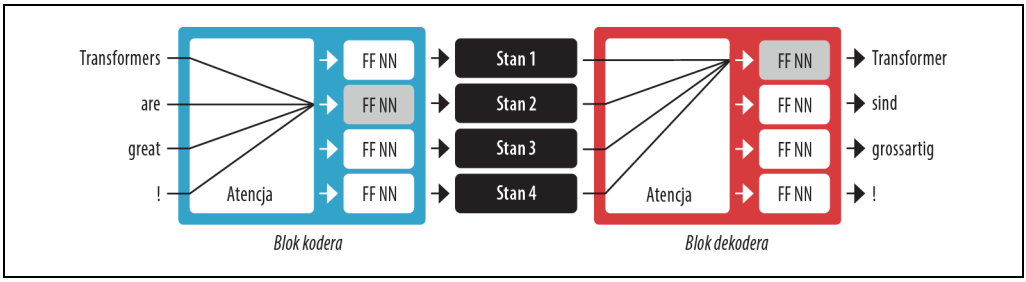
Skupiając się na tym, które tokeny wejściowe są najistotniejsze w każdym kroku czasowym, modele oparte na atencji mogą się uczyć niebanalnych związków między słowami w wygenerowanym tłumaczeniu i w zdaniu źródłowym. Na przykład na rysunku 1.5 pokazano wagi atencji dla modelu tłumaczenia z angielskiego na francuski, przy czym każdy piksel oznacza wagę. Rysunek pokazuje, jak dekoder poprawnie dopasowuje słowa „zone" i „Area", które mają inną kolejność w dwóch językach.

Choć atencja znacznie poprawiła jakość tłumaczeń, używanie modeli rekurencyjnych w charakterze kodera i dekodera nadal miało poważną wadę: obliczenia są z natury sekwencyjne, co nie pozwala przetwarzać sekwencji wejściowej w sposób równoległy.

Wprowadzenie transformerów przyniosło nowy paradygmat modelowania: całkowicie pozbywamy się rekurencji, zamiast tego w pełni polegając na specjalnej formie atencji zwanej **samouwagą** (ang. *self-attention*). Samouwagę opiszemy dokładniej w rozdziale 3., ale podstawowy pomysł jest taki, aby umożliwić atencji operowanie na wszystkich stanach w tej samej warstwie sieci neuronowej. Pokazano to na rysunku 1.6, na którym zarówno koder, jak i dekoder mają własne mechanizmy samouwagi, których dane wyjściowe są przekazywane do sieci neuronowych z propagacją w przód (ang. *feed-forward neural network*, FF NN). Architektura ta uczy się znacznie szybciej niż modele rekurencyjne i utorowała drogę do wielu niedawnych przełomów w NLP.



Rysunek 1.5. Dopasowywanie słów angielskich do wygenerowanego francuskiego tłumaczenia przez koder-dekoder RNN (dzięki uprzejmości Dzmitry'ego Bahdanaua)



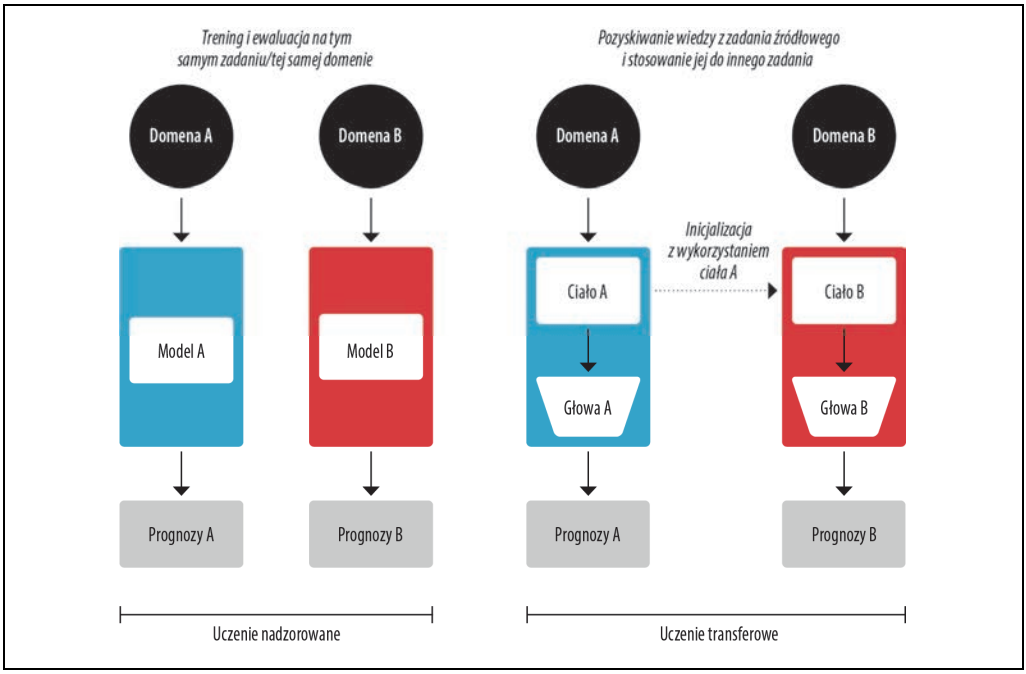
Rysunek 1.6. Architektura koder-dekoder pierwotnego transformera

W pierwotnym projekcie transformera model tłumaczeniowy wytrenowano od zera na dużym korpusie par zdań w różnych językach. Jednak w wielu praktycznych zastosowaniach NLP nie mamy dostępu do dużej ilości opatrzonych etykietami danych tekstowych, na których moglibyśmy wytrenować nasz model. Do rozpoczęcia rewolucji transformerów potrzebny był ostatni kawałek układanki: uczenie transferowe.

Uczenie transferowe w NLP

We współczesnych systemach wizji komputerowej często używa się uczenia transferowego, aby wytrenować konwolucyjną sieć neuronową, taką jak ResNet, na jednym zadaniu, a następnie zaadaptować ją lub też **dostroić** do nowego zadania. Dzięki temu sieć może wykorzystać wiedzę zdobytą podczas pierwotnego zadania. Z architektonicznego punktu widzenia polega to na podzieleniu modelu na **ciało** i **głowę**, gdzie głowa jest siecią dostosowaną do konkretnego zadania. Podczas treningu

wagi ciała uczą się ogólnych cech domeny źródłowej, a wag tych używa się od zainicjalizowania nowego modelu do nowego zadania[7]. W porównaniu z tradycyjnym uczeniem nadzorowanym podejście to zwykle generuje modele wysokiej jakości, które można znacznie efektywniej trenować na konkretnych zadaniach, używając znacznie mniejszej ilości danych opatrzonych etykietami. Porównanie obu podejść pokazano na rysunku 1.7.



Rysunek 1.7. Porównanie tradycyjnego uczenia nadzorowanego (po lewej stronie) i uczenia transferowego (po prawej)

W systemach wizji komputerowej modele najpierw trenuje się na dużych zbiorach danych, takich jak ImageNet (*https://image-net.org*), które zawierają miliony obrazów. Proces ten nosi nazwę **treningu wstępnego**, a jego celem jest nauczenie modelu podstawowych cech obrazów, takich jak krawędzie lub kolory. Wstępnie wytrenowane modele można następnie dostroić do konkretnego zadania, na przykład klasyfikowania gatunków kwiatów, przy użyciu względnie niewielkiej liczby przykładów opatrzonych etykietami (zwykle po kilkaset na klasę). Modele dostrojone zazwyczaj osiągają wyższą dokładność niż modele nadzorowane wytrenowane od podstaw na tej samej ilości danych opatrzonych etykietami.

Choć uczenie transferowe stało się standardowym podejściem w wizji komputerowej, przez wiele lat nie było jasne, jak mógłby wyglądać analogiczny proces szkolenia wstępnego w NLP. W rezultacie aplikacje NLP zwykle wymagały dużej ilości danych opatrzonych etykietami do osiągnięcia przyzwoitych wyników. A nawet wówczas ich trafność nie dorównywała temu, co osiągnięto w dziedzinie wizji.

[7] Wagi to parametry sieci neuronowej, które zmieniają się w wyniku uczenia.

W latach 2017 i 2018 kilka grup badawczych zaproponowało nowe podejścia, które wreszcie pozwoliły wykorzystać uczenie transferowe w NLP. Zaczęło się od spostrzeżeń badaczy z OpenAI, którzy uzyskali dobre wyniki w zadaniu klasyfikacji poprzez użycie cech wyodrębnionych z nienadzorowanego treningu wstępnego[8]. Następnie opracowano metodę ULMFiT, która wprowadziła ogólne ramy adaptowania wstępnie wytrenowanych modeli LSTM do różnych zadań[9].

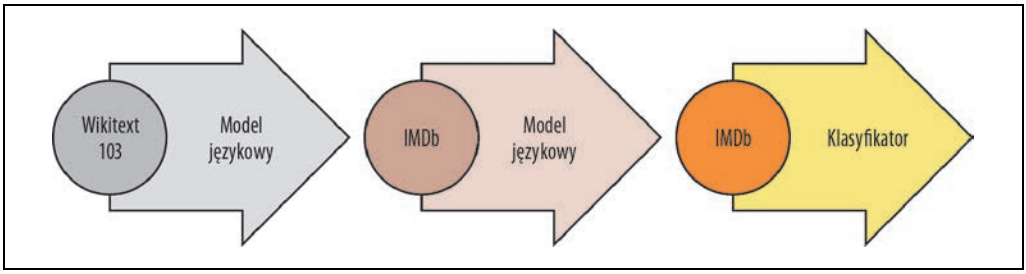
Jak pokazano na rysunku 1.8, metoda ULMFiT obejmuje trzy główne etapy:

Trening wstępny

Cel treningu wstępnego jest prosty: przewidywać następne słowo na podstawie poprzednich słów. Zadanie to określa się mianem **modelowania językowego**. Elegancja tego podejścia kryje się w tym, że niepotrzebne są żadne dane opatrzone etykietami i że można wykorzystać obfite, łatwo dostępne źródła tekstu, takie jak Wikipedia[10].

Adaptacja dziedzinowa

Po wstępnym wytrenowaniu modelu językowego na dużym korpusie następnym etapem jest zaadaptowanie go do korpusu dziedzinowego (na przykład z Wikipedii do korpusu recenzji filmowych IMDb, jak pokazano na rysunku 1.8). W tym etapie nadal używa się modelowania językowego, ale teraz model musi przewidywać następne słowo w docelowym korpusie.



Rysunek 1.8. Proces ULMFiT (dzięki uprzejmości Jeremy'ego Howarda)

Dostrajanie

W tym etapie model językowy dostraja się z wykorzystaniem warstwy klasyfikacji specyficznej dla docelowego zadania (na przykład klasyfikowania odczuć w recenzjach filmowych z rysunku 1.8).

Poprzez wprowadzenie praktycznych sposobów treningu wstępnego i uczenia transferowego do NLP projekt ULMFiT zapewnił brakujący kawałek układanki, umożliwiając powstanie transformerów. W 2018 r. wydano dwa transformery, które łączyły samouwagę z uczeniem transferowym:

[8] A. Radford, R. Jozefowicz, I. Sutskever, *Learning to Generate Reviews and Discovering Sentiment* (*https://arxiv.org/abs/1704.01444*), 2017.

[9] Powiązana praca ELMo (ang. *Embeddings from Language Models*) pokazała, jak wstępny trening sieci LSTM pozwala uzyskać wysokiej jakości osadzenia słów do konkretnych zadań.

[10] Dotyczy to w większej mierze języka angielskiego niż większości innych języków świata, dla których pozyskanie dużego korpusu cyfrowego tekstu może być trudne. Szukanie sposobów na wypełnienie tej luki jest aktywnym obszarem badań i aktywizmu NLP.

GPT

Używa tylko dekoderowej części architektury transformera oraz tego samego podejścia do modelowania językowego co ULMFiT. GPT wstępnie wytrenowano na zbiorze BookCorpus[11], który składa się z 7000 nieopublikowanych książek z różnych gatunków, takich jak przygoda, fantastyka i romans.

BERT

Używa ekonderowej części architektury transformera i specjalnej odmiany modelowania językowego nazywanej **maskowanym modelowaniem językowym**. Celem modelowania maskowanego jest przewidywanie losowo zamaskowanych słów w tekście. Na przykład otrzymawszy zdanie „I looked at my [MASKA] and saw that [MASKA] was late.", model ma przewidzieć najbardziej prawdopodobne słowa oznaczone przez [MASKA]. Model BERT wstępnie wytrenowano na zbiorze BookCorpus i angielskiej Wikipedii.

GPT i BERT wyznaczyły nowe standardy w wielu różnych zadaniach NLP i zapoczątkowały erę transformerów.

Ponieważ jednak różne laboratoria badawcze wydawały swoje modele na niekompatybilnych platformach (PyTorch lub TensorFlow), praktycy NLP nie zawsze mogli łatwo dostosować te modele do własnych aplikacji. Wydanie biblioteki 🤗 Transformers (*https://oreil.ly/Z79jF*) umożliwiło stopniowe zbudowanie zunifikowanego interfejsu API obsługującego ponad 50 architektur. Biblioteka ta stała się katalizatorem rozwoju badań nad transformerami i szybko trafiła w ręce praktyków NLP, ułatwiając integrowanie modeli z wieloma rzeczywistymi aplikacjami. Przyjrzyjmy się jej bliżej!

Hugging Face Transformers — eliminowanie luki

Stosowanie nowatorskiej architektury uczenia maszynowego do nowego zadania bywa skomplikowanym przedsięwzięciem i zwykle składa się z następujących etapów:

1. Implementowanie architektury modelu w kodzie, najczęściej z wykorzystaniem PyTorch lub TensorFlow.
2. Wczytanie wstępnie wytrenowanych wag (jeśli są dostępne) z serwera.
3. Wstępne przetworzenie danych wejściowych, przepuszczenie ich przez model i zastosowanie jakiegoś przetwarzania końcowego specyficznego dla zadania.
4. Zaimplementowanie modułów wczytywania danych oraz definiowanie funkcji straty i optymalizatorów w celu wytrenowania modelu.

Każdy z tych etapów wymaga niestandardowej logiki dla każdego modelu i zadania. Tradycyjnie (choć nie zawsze!), kiedy grupa badawcza publikuje nowy artykuł, to publikuje również kod wraz z wagami modelu. Kod ten jest jednak rzadko ustandaryzowany, a adaptowanie go do nowych zastosowań często zajmuje wiele dni.

[11] Y. Zhu et al., *Aligning Books and Movies: Towards Story-like Visual Explanations by Watching Movies and Reading Books* (*https://arxiv.org/abs/1506.06724*), 2015.

Z odsieczą przychodzi biblioteka 🤗 Transformers! Zapewnia ona standardowy interfejs do szerokiej gamy modeli transformerów, a także kod i narzędzia, które pozwalają adaptować te modele do nowych zastosowań. Biblioteka obecnie obsługuje trzy najpopularniejsze platformy uczenia głębokiego (PyTorch, TensorFlow i JAX) i pozwala łatwo przełączać się między nimi. Ponadto oferuje głowy dostosowane do konkretnych zadań, co pozwala łatwo dostrajać transformery do zadań pochodnych, takich jak klasyfikacja tekstu, rozpoznawanie nazwanych encji i odpowiadanie na pytania. Skraca to czas, który praktycy spędzają na trenowaniu i testowaniu modeli, z tygodnia do jednego popołudnia!

Przekonasz się o tym w następnym podrozdziale, w którym pokażemy, że wystarczy kilka wierszy kodu, aby zastosować 🤗 Transformers do niektórych typowych zadań NLP, z którymi prawdopodobnie zetkniesz się w rzeczywistym świecie.

Przegląd zastosowań transformerów

Każde zadanie NLP zaczyna się od fragmentu tekstu, takiego jak poniższa zmyślona opinia klienta o pewnym zamówieniu internetowym:

```
text = """Dear Amazon, last week I ordered an Optimus Prime action figure from your online
store in Germany. Unfortunately, when I opened the package, I discovered to my horror that
I had been sent an action figure of Megatron instead! As a lifelong enemy of the
Decepticons, I hope you can understand my dilemma. To resolve the issue, I demand an
exchange of Megatron for the Optimus Prime figure I ordered. Enclosed are copies of my
records concerning this purchase. I expect to hear from you soon. Sincerely, Bumblebee."""
```

W zależności od aplikacji tekst, z którym pracujesz, może być umową prawną, opisem produktu albo czymś zupełnie innym. W przypadku opinii klienta zapewne chciałbyś wiedzieć, czy jest ona pozytywna, czy negatywna. Zadanie to jest znane jako **analiza odczuć** i stanowi część szerszego tematu **klasyfikacji tekstu**, który zbadamy w rozdziale 2. Na razie sprawdźmy, co trzeba zrobić, żeby za pomocą biblioteki 🤗 Transformers określić odczucia wyrażone w powyższym fragmencie tekstu.

Klasyfikacja tekstu

Jak dowiesz się w późniejszych rozdziałach, biblioteka 🤗 Transformers ma wielowarstwowy interfejs API, który pozwala używać jej na różnych poziomach abstrakcji. W tym rozdziale zaczniemy od **potoków**, które ukrywają wszystkie czynności niezbędne do przekształcenia „surowego” tekstu w prognozy dostrojonego modelu.

W bibliotece 🤗 Transformers tworzymy egzemplarz potoku poprzez wywołanie funkcji `pipeline()` i podanie nazwy zadania, którym jesteśmy zainteresowani:

```
from transformers import pipeline
classifier = pipeline("text-classification")
```

Kiedy uruchomisz ten kod po raz pierwszy, zobaczysz kilka pasków postępu, ponieważ potok automatycznie pobierze wagi modelu z witryny Hugging Face Hub (*https://oreil.ly/zLK11*). Kiedy utworzysz potok po raz drugi, biblioteka wykryje, że pobrałeś już wagi, i użyje lokalnie

przechowywanej wersji. Potok `text-classification` domyślnie używa modelu przeznaczonego do analizy odczuć, ale obsługuje również klasyfikację wieloklasową i wieloetykietową.

Teraz, kiedy mamy już nasz potok, wygenerujmy jakieś prognozy! Każdy potok przyjmuje na wejściu łańcuch tekstu (albo listę łańcuchów) i zwraca listę prognoz. Każda prognoza jest słownikiem Pythona, więc używamy biblioteki Pandas, aby wyświetlić ją jako ramkę danych (`DataFrame`):

```
import pandas as pd
outputs = classifier(text)
pd.DataFrame(outputs)
```

	label	score
0	NEGATIVE	0.901546

W tym przypadku model jest bardzo „pewny", że uczucia wyrażone w tekście są negatywne, co ma sens, skoro mamy do czynienia ze skargą zdenerwowanego klienta! Zauważ, że w zadaniu analizy odczuć potok zwraca tylko jedną z etykiet, `POSITIVE` lub `NEGATIVE`, ponieważ wartość drugiej można ustalić poprzez obliczenie różnicy `1-score`.

Przyjrzyjmy się teraz innemu typowemu zadaniu — identyfikowaniu nazwanych encji w tekście.

Rozpoznawanie nazwanych encji

Przewidywanie odczuć wyrażonych w opinii klienta jest dobrym pierwszym krokiem, ale często chcemy wiedzieć, czy opinia dotyczyła konkretnego przedmiotu lub usługi. W NLP rzeczywiste obiekty, takie jak produkty, miejsca i ludzie, określa się mianem **nazwanych encji**, a proces wyodrębniania ich z tekstu to rozpoznawanie nazwanych encji (ang. *named entity recognition*, NER). Aby zastosować NER, wczytujemy odpowiedni potok i przekazujemy do niego naszą opinię klienta:

```
ner_tagger = pipeline("ner", aggregation_strategy="simple")
outputs = ner_tagger(text)
pd.DataFrame(outputs)
```

	entity_group	score	word	start	end
0	ORG	0.879010	Amazon	5	11
1	MISC	0.990859	Optimus Prime	36	49
2	LOC	0.999755	Germany	90	97
3	MISC	0.556569	Mega	208	212
4	PER	0.590256	##tron	212	216
5	ORG	0.669692	Decept	253	259
6	MISC	0.498350	##icons	259	264
7	MISC	0.775361	Megatron	350	358
8	MISC	0.987854	Optimus Prime	367	380
9	PER	0.812096	Bumblebee	502	511

Jak widzisz, potok wykrył wszystkie encje i przypisał im również kategorie, takie jak `ORG` (organizacja), `LOC` (miejsce) lub `PER` (osoba). Użyliśmy argumentu `aggregation_strategy`, aby pogrupować słowa według prognoz modelu. Na przykład encja „Optimus Prime" składa się z dwóch słów, ale przypisano jej pojedynczą kategorię: `MISC` (różne). Wartości `score` informują, z jaką pewnością model zidentyfikował poszczególne encje. Widzimy, że jest najmniej pewny słowa „Decepticons" i pierwszego wystąpienia słowa „Megatron", których nie zdołał zgrupować jako pojedynczych encji.

Czy zauważyłeś dziwne znaki krzyżyka (`#`) w kolumnie `word` powyższej tabeli? Są one generowane przez **tokenizator** modelu, który rozkłada słowa na niepodzielne jednostki zwane **tokenami**. Tokenizacja zostanie opisana w rozdziale 2.

Wyodrębnianie wszystkich nazwanych encji jest bez wątpienia użyteczne, ale czasem chcielibyśmy zadawać bardziej ukierunkowane pytania. Możemy wówczas użyć **odpowiadania na pytania**.

Odpowiadanie na pytania

W odpowiadaniu na pytania przekazujemy modelowi fragment tekstu zwany **kontekstem** wraz z pytaniem, na które chcemy uzyskać odpowiedź. Model zwraca zakres tekstu odpowiadający odpowiedzi. Zobaczmy, co otrzymamy, gdy zadamy konkretne pytanie dotyczące naszej opinii klienta:

```
reader = pipeline("question-answering")
question = "What does the customer want?"
outputs = reader(question=question, context=text)
pd.DataFrame([outputs])
```

	score	start	end	answer
0	0.631291	335	358	an exchange of Megatron

Jak widać, oprócz odpowiedzi potok zwraca również początkową i końcową wartość całkowitą. Odpowiadają one indeksom znakowym, pod którymi znaleziono odpowiedź (podobnie jak w przypadku tagowania NER). Istnieje kilka odmian odpowiadania na pytania, które zbadamy w rozdziale 7., ale ta konkretna odmiana jest nazywana **ekstrakcyjnym odpowiadaniem na pytania**, ponieważ odpowiedź „ekstrahuje" się bezpośrednio z tekstu.

Technika ta pozwala szybko wyodrębnić istotne informacje z opinii klienta. Co jednak robić, jeśli otrzymasz mnóstwo rozwlekłych zażaleń i nie będziesz mieć czasu, żeby przeczytać je wszystkie? Zobaczmy, czy w takiej sytuacji pomoże Ci model streszczania!

Streszczanie

Celem streszczania tekstu jest przyjęcie długiego tekstu na wejściu i wygenerowanie jego krótkiej wersji z zachowaniem wszystkich istotnych informacji. Jest to zadanie znacznie bardziej skomplikowane od poprzednich, ponieważ wymaga, aby model generował spójny logicznie tekst. Potok streszczania możemy utworzyć w poniższy, zapewne znajomy już sposób:

```
summarizer = pipeline("summarization")
outputs = summarizer(text, max_length=45, clean_up_tokenization_spaces=True)
```

```
print(outputs[0]['summary_text'])
 Bumblebee ordered an Optimus Prime action figure from your online store in Germany.
Unfortunately, when I opened the package, I discovered to my horror that I had been sent
an action figure of Megatron instead.
```

To streszczenie jest całkiem przyzwoite! Choć zawiera skopiowane fragmenty oryginalnego tekstu, model zdołał uchwycić sedno problemu i poprawnie określił, że skargę napisał „Bumblebee" (które to imię pojawiło się na końcu). W tym przykładzie widać również, że przekazaliśmy potokowi pewne argumenty, takie jak `max_length` i `clean_up_tokenization_spaces`; pozwalają one wpływać na wyniki w czasie działania modelu.

Co jednak zrobić, jeśli otrzymasz opinię w języku, którego nie znasz? Mógłbyś użyć usługi Google Translate albo przetłumaczyć tekst za pomocą własnego transformera!

Tłumaczenie

Podobnie jak streszczanie, tłumaczenie jest zadaniem, którego wynikiem jest wygenerowany tekst. Użyjmy potoku tłumaczenia, aby przełożyć angielski tekst na język niemiecki:

```
translator = pipeline("translation_en_to_de", model="Helsinki-NLP/opus-mt-en-de")
outputs = translator(text, clean_up_tokenization_spaces=True, min_length=100)
print(outputs[0]['translation_text'])

Sehr geehrter Amazon, letzte Woche habe ich eine Optimus Prime Action Figur aus
Ihrem Online-Shop in Deutschland bestellt. Leider, als ich das Paket offnete,
entdeckte ich zu meinem Entsetzen, dass ich stattdessen eine Action Figur von
Megatron geschickt worden war! Als lebenslanger Feind der Decepticons, Ich
hoffe, Sie konnen mein Dilemma verstehen. Um das Problem zu losen, Ich fordere
einen Austausch von Megatron fur die Optimus Prime Figur habe ich bestellt.
Anbei sind Kopien meiner Aufzeichnungen uber diesen Kauf. Ich erwarte, bald von
Ihnen zu horen. Aufrichtig, Bumblebee.
```

Ponownie, model utworzył bardzo dobre tłumaczenie, które poprawnie używa formalnych niemieckich zaimków, takich jak „Ihrem" i „Sie". Pokazaliśmy też, jak zmienić domyślny model w potoku, aby wybrać ten, który najlepiej pasuje do danego zastosowania — a w witrynie Hugging Face Hub można znaleźć modele dla tysięcy par językowych. Zanim cofniemy się o krok i przyjrzymy całemu ekosystemowi Hugging Face, zbadajmy jeszcze jedno zastosowanie.

Generowanie tekstu

Przypuśćmy, że chcielibyśmy szybciej odpowiadać na opinie klientów, w czym mogłaby pomóc funkcja autouzupełniania. Za pomocą modelu generowania tekstu możemy zrobić to w następujący sposób:

```
generator = pipeline("text-generation")
response = "Dear Bumblebee, I am sorry to hear that your order was mixed up."
prompt = text + "\n\nCustomer service response:\n" + response
outputs = generator(prompt, max_length=200)
print(outputs[0]['generated_text'])

Dear Amazon, last week I ordered an Optimus Prime action figure from your online
store in Germany. Unfortunately, when I opened the package, I discovered to my
```

```
horror that I had been sent an action figure of Megatron instead! As a lifelong
enemy of the Decepticons, I hope you can understand my dilemma. To resolve the
issue, I demand an exchange of Megatron for the Optimus Prime figure I ordered.
Enclosed are copies of my records concerning this purchase. I expect to hear
from you soon. Sincerely, Bumblebee.

Customer service response:
Dear Bumblebee, I am sorry to hear that your order was mixed up. The order was
completely mislabeled, which is very common in our online store, but I can
appreciate it because it was my understanding from this site and our customer
service of the previous day that your order was not made correct in our mind and
that we are in a process of resolving this matter. We can assure you that your
order
```

Cóż, prawdopodobnie nie użylibyśmy tego autouzupełnienia, aby uspokoić Bumblebee, ale wiesz już, na czym to mniej więcej polega.

Teraz, kiedy poznałeś już kilka interesujących zastosowań modeli transformerów, zastanawiasz się zapewne, gdzie odbywa się ich trening. Wszystkie modele, których użyliśmy w tym rozdziale, są publicznie dostępne i już dostrojone do konkretnych zadań. Jednak ogólnie rzecz biorąc, będziesz chciał dostrajać modele do swoich własnych danych, a w następnych rozdziałach pokażemy Ci, jak to zrobić.

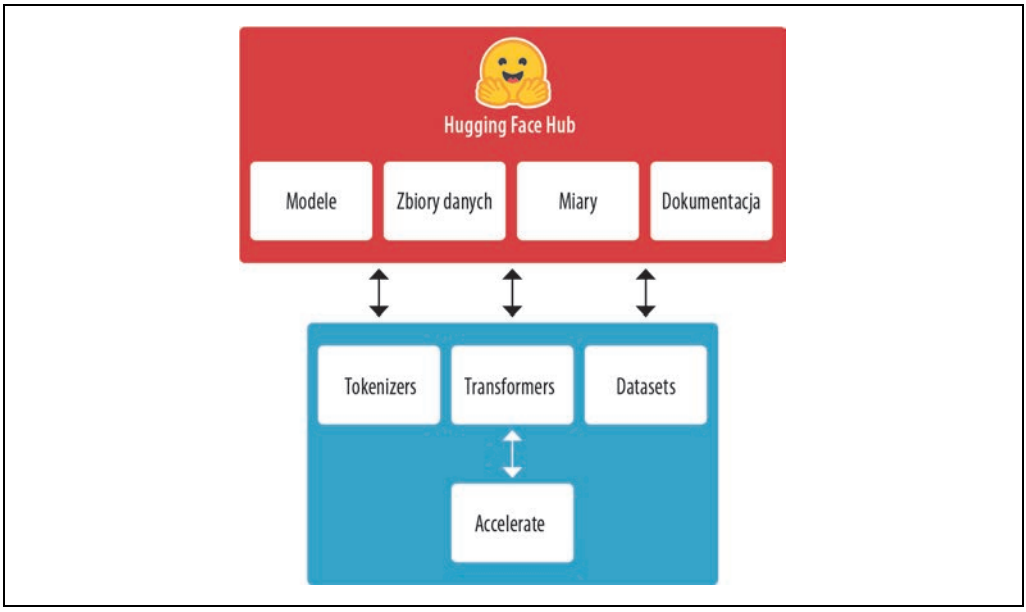
Jednak trenowanie modelu jest tylko niewielką częścią dowolnego projektu NLP — inne kluczowe komponenty to metody efektywnego przetwarzania danych, możliwość udostępniania wyników współpracownikom oraz odtwarzalność Twojej pracy. Na szczęście biblioteka 🤗 Transformers jest otoczona dużym ekosystemem przydatnych narzędzi, które wspierają nowoczesne procesy uczenia maszynowego. Przyjrzyjmy się im teraz.

Ekosystem Hugging Face

To, co zaczęło się od biblioteki 🤗 Transformers, szybko obrosło całym ekosystemem bibliotek i narzędzi, które przyspieszają projekty NLP i uczenia maszynowego. Ekosystem Hugging Face składa się z dwóch głównych części: rodziny bibliotek oraz witryny Hub, jak pokazano na rysunku 1.9. Biblioteki zawierają kod, a Hub udostępnia wstępnie wytrenowane wagi modeli, zbiory danych, skrypty ewaluacyjne i znacznie więcej. W tym podrozdziale krótko opiszemy poszczególne komponenty. Pominiemy samą bibliotekę 🤗 Transformers, ponieważ już ją omówiliśmy i będziemy do niej wielokrotnie wracać w dalszych rozdziałach książki.

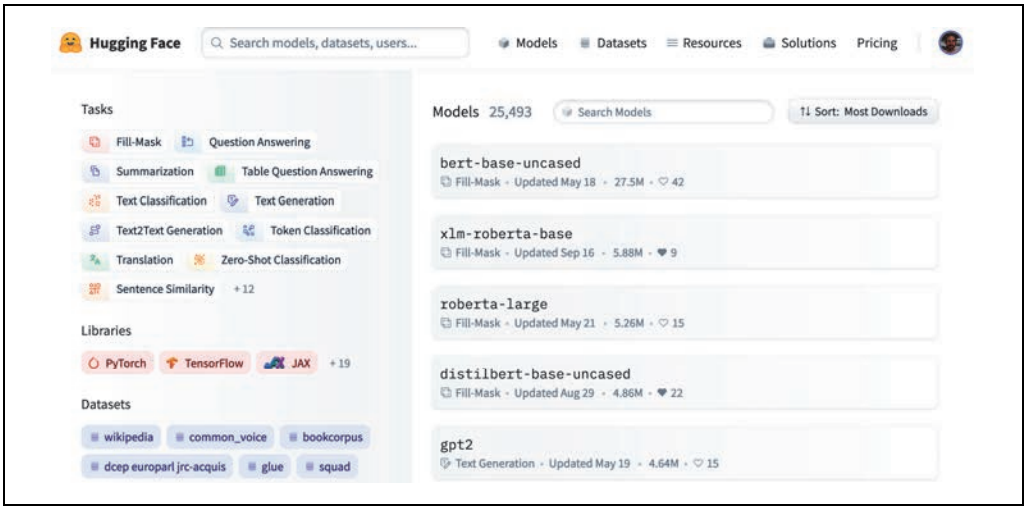
Hugging Face Hub

Jak wspomniano wcześniej, uczenie transferowe jest jednym z kluczowych czynników sukcesu transformerów, ponieważ umożliwia ponowne wykorzystanie wstępnie wytrenowanych modeli do nowych zadań. W rezultacie bardzo ważna jest możliwość szybkiego wczytywania wstępnie wytrenowanych modeli i eksperymentowania z nimi.



Rysunek 1.9. Przegląd ekosystemu Hugging Face

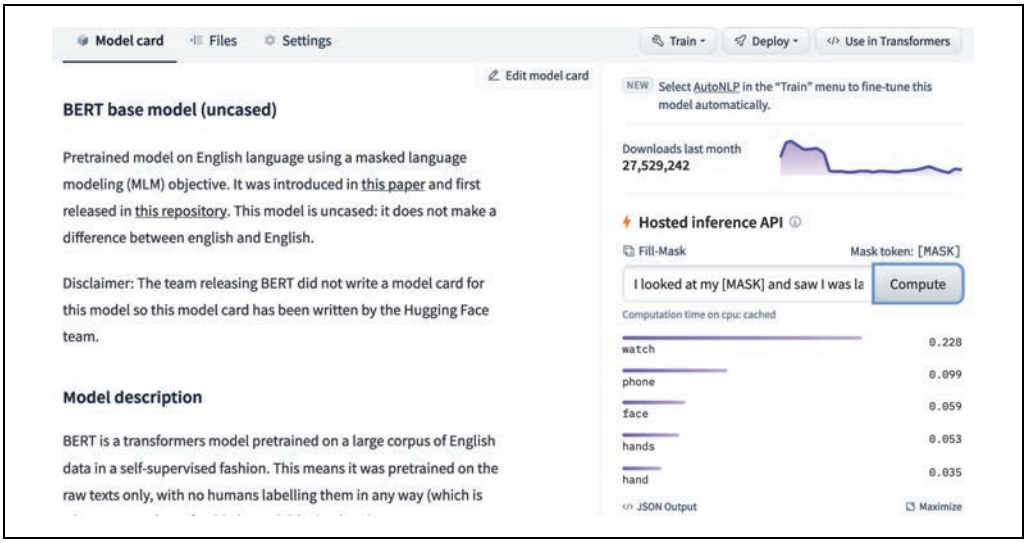
Repozytorium Hugging Face Hub zawiera ponad 20 000 bezpłatnych modeli. Jak pokazano na rysunku 1.10, dostępne są filtry zadań, platform, zbiorów danych itd., które pomagają nawigować po repozytorium i szybko znajdować obiecujących kandydatów. Jak widziałeś w przykładach z potokami, wczytanie obiecującego modelu do własnego kodu zajmuje dosłownie jeden wiersz. Upraszcza to eksperymentowanie z szeroką gamą modeli i pozwala skupić się na tych częściach projektu, które są specyficzne dla dziedziny.



Rysunek 1.10. Strona Models witryny Hugging Face Hub wyświetlająca filtry po lewej stronie i listę modeli po prawej stronie

Oprócz wag modeli Hub przechowuje również zbiory danych i skrypty do obliczania miar, które pozwalają odtwarzać opublikowane wyniki lub wykorzystać dodatkowe dane we własnej aplikacji.

Hub oferuje również **karty modeli i zestawów danych**, które dokumentują zawartość modeli i zbiorów danych oraz pomagają podjąć decyzję, czy będą one odpowiednie dla Ciebie. Jedną z najprzydatniejszych funkcji witryny jest możliwość bezpośredniego wypróbowania dowolnego modelu za pomocą różnych interaktywnych „widgetów", jak pokazano na rysunku 1.11.



Rysunek 1.11. Przykładowa karta modelu w witrynie Hugging Face Hub; widget inferencyjny, który umożliwia interakcję z modelem, znajduje się po prawej stronie

Kontynuując nasz przegląd, przejdźmy teraz do biblioteki 🤗 Tokenizers.

PyTorch (*https://oreil.ly/AyTYC*) i TensorFlow (*https://oreil.ly/JOKgq*) oferują własne repozytoria. Warto je sprawdzić, jeśli określony model lub zbiór danych nie jest dostępny w witrynie Hugging Face Hub.

Hugging Face Tokenizers

Każdy przykładowy potok, który widziałeś w tym rozdziale, zawiera etap tokenizacji, w którym surowy tekst jest dzielony na mniejsze części zwane tokenami. Szczegółowy opis tego procesu znajdziesz w rozdziale 2., a na razie wystarczy wiedzieć, że tokeny mogą być słowami, częściami słów albo po prostu znakami, takimi jak znaki interpunkcyjne. Modele transformerów trenuje się na liczbowych reprezentacjach tych tokenów, więc poprawna realizacja tego etapu ma duży wpływ na cały projekt NLP!

Biblioteka 🤗 Tokenizers (*https://github.com/huggingface/tokenizers*) oferuje wiele strategii tokenizacji i niezwykle szybko tokenizuje tekst dzięki „zapleczu" napisanemu w Ruście[12]. Przeprowadza też wszystkie etapy przetwarzania wstępnego i końcowego, takie jak normalizacja danych wejściowych i przekształcanie wyników modelu w wymagany format. Za pomocą biblioteki 🤗 Tokenizers możemy wczytać tokenizator w taki sam sposób, w jaki wczytujemy wstępnie wytrenowane wagi modeli w bibliotece 🤗 Transformers.

Aby trenować i oceniać modele, potrzebujemy zbioru danych i miar, więc przyjrzyjmy się bibliotece 🤗 Datasets, która odpowiada za ten aspekt.

Hugging Face Datasets

Wczytywanie, przetwarzanie i zapisywane zbiorów danych to uciążliwy proces, zwłaszcza gdy zbiór danych staje się za duży, żeby zmieścić się w pamięci RAM laptopa. Ponadto zazwyczaj trzeba napisać różne skrypty, aby pobrać dane i przekształcić je do standardowego formatu.

Biblioteka 🤗 Datasets (*https://oreil.ly/959YT*) upraszcza ten proces, udostępniając standardowy interfejs do tysięcy zbiorów danych w witrynie Hub (*https://oreil.ly/Rdhcu*). Oferuje też inteligentną pamięć podręczną (żebyś nie musiał od nowa przeprowadzać przetwarzania wstępnego za każdym razem, kiedy uruchamiasz swój kod) i pomaga uniknąć ograniczeń związanych z pamięcią RAM poprzez wykorzystanie specjalnego mechanizmu nazywanego **mapowaniem pamięci**, który przechowuje zawartość pliku w pamięci wirtualnej i umożliwia wielu procesom efektywniejsze modyfikowanie pliku. Biblioteka jest też kompatybilna z popularnymi platformami, takimi jak Pandas i NumPy, więc możesz korzystać z tych narzędzi do przetwarzania danych, z którymi czujesz się najbardziej komfortowo.

Dobry zbiór danych i zaawansowany model nie zdadzą się jednak na nic, jeśli nie będziesz mógł wiarygodnie zmierzyć wydajności. Niestety, klasyczne miary NLP mają wiele różnych implementacji, które mogą nieco się różnić i prowadzić do zwodniczych wyników. Oferując skrypty dla wielu miar, biblioteka 🤗 Datasets pomaga zwiększyć powtarzalność eksperymentów i wiarygodność wyników.

Dysponując bibliotekami 🤗 Transformers, 🤗 Tokenizers i 🤗 Datasets, mamy wszystko, czego potrzebujemy, aby trenować nasze własne modele transformerów! Jak jednak przekonamy się w rozdziale 10., w niektórych sytuacjach potrzebujemy precyzyjnej kontroli nad pętlą treningową. Tu na scenę wkracza ostatnia biblioteka ekosystemu: 🤗 Accelerate.

Hugging Face Accelerate

Jeśli kiedykolwiek napisałeś własny skrypt treningowy w PyTorch, istnieje duże prawdopodobieństwo, że nabawiłeś się bólu głowy, próbując przenieść kod działający w laptopie do klastra serwerów Twojej organizacji. Biblioteka 🤗 Accelerate (*https://oreil.ly/iRfDe*) dodaje do zwykłych pętli treningowych warstwę abstrakcji, która dba o całą niestandardową logikę potrzebną w infrastrukturze treningowej. Dosłownie przyspiesza ona pracę, upraszczając zmianę infrastruktury, jeśli okaże się to konieczne.

[12] Rust (*https://rust-lang.org*) to wysokowydajny język programowania.

To już wszystkie podstawowe elementy ekosystemu Hugging Face. Zanim jednak zakończymy ten rozdział, przyjrzyjmy się kilku typowym wyzwaniom, jakie wiążą się z próbami wdrożenia transformerów w rzeczywistym świecie.

Główne wyzwania związane z transformerami

W tym rozdziale rzuciliśmy okiem na szeroki zakres zadań NLP, które można rozwiązać za pomocą modeli transformerów. Kiedy czytamy nagłówki w mediach, czasami może się nam wydawać, że ich możliwości są nieograniczone. Jednak pomimo swojej przydatności transformery nie są panaceum na każdy problem. Oto kilka związanych z nimi wyzwań, które zbadamy w niniejszej książce:

Język

Badania nad NLP są zdominowane przez język angielski. Istnieje kilka modeli dla innych języków, ale trudno jest znaleźć wstępnie wytrenowane modele dla języków rzadkich lub mających niewiele zasobów tekstowych. W rozdziale 4. zbadamy transformery wielojęzyczne i ich zdolność do wykonywania transferu międzyjęzykowego metodą *zero-shot*, czyli bez dostępnych przykładów z docelowego języka.

Dostępność danych

Choć możemy użyć uczenia transferowego, aby radykalnie ograniczyć ilość opatrzonych etykietami danych treningowych, których potrzebują nasze modele, to wciąż jest to znacznie więcej danych w porównaniu z tym, ile potrzebowałby człowiek. Radzenie sobie w scenariuszach, w których mamy mało opatrzonych etykietami danych albo nie mamy ich w ogóle, jest tematem rozdziału 9.

Praca z długimi dokumentami

Samouwaga działa doskonale na tekstach o długości akapitu, ale staje się bardzo kosztowna, kiedy przechodzimy do dłuższych tekstów, takich jak całe dokumenty. Podejścia, które pomagają złagodzić ten problem, zostaną omówione w rozdziale 11.

Nieprzezroczystość

Podobnie jak inne modele uczenia głębokiego, transformery są w dużym stopniu „nieprzezroczyste". Trudno lub nie da się stwierdzić, „dlaczego" model dokonał konkretnej prognozy. Jest to szczególnie poważny problem, kiedy modele te wykorzystuje się do podejmowania krytycznych decyzji. Niektóre sposoby sondowania błędów modeli transformerów zbadamy w rozdziałach 2. i 4.

Uprzedzenia

Modele transformerów trenuje się głównie na danych dostępnych w internecie. Oznacza to, że wszystkie uprzedzenia obecne w danych zostają „wdrukowane" w model. Upewnienie się, że dane nie są rasistowskie albo seksistowskie, jest trudnym zadaniem. Niektóre z tych problemów omówimy szczegółowo w rozdziale 10.

Choć powyższe wyzwania są niełatwe, wiele z nich można przezwyciężyć. Omówimy je nie tylko we wspomnianych wyżej rozdziałach, ale również w niemal każdym kolejnym rozdziale książki.

Podsumowanie

Mamy nadzieję, że nie możesz się już doczekać, żeby zacząć trenować te wszechstronne modele i integrować je z Twoimi własnymi aplikacjami! W tym rozdziale przekonałeś się, że za pomocą zaledwie kilku wierszy kodu możesz wykorzystać najnowocześniejsze modele w takich zadaniach jak klasyfikacja, rozpoznawanie nazwanych encji, odpowiadanie na pytania, tłumaczenie i streszczanie, choć tak naprawdę jest to tylko wierzchołek góry lodowej.

W następnych rozdziałach dowiesz się, jak adaptować transformery do szerokiej gamy zastosowań, takich jak budowanie klasyfikatora tekstu albo lekkiego modelu produkcyjnego, a nawet trenowanie modelu językowego od podstaw. Przyjmiemy podejście praktyczne, co oznacza, że każdej omawianej koncepcji będzie towarzyszył kod, który będziesz mógł uruchomić w witrynie Google Colab albo na własnym komputerze z procesorem graficznym.

Teraz, kiedy znasz podstawowe pojęcia związane z transformerami, czas ubrudzić ręce i stworzyć pierwszą aplikację: klasyfikator tekstu. Jest to temat następnego rozdziału!

Wybitna książka poświęcona wybitnej bibliotece — wzór przejrzystości!

Jeremy Howard, profesor Uniwersytetu Queensland

Modele transformacyjne zmieniły sposób przetwarzania języka naturalnego. Rewolucja rozpoczęła się w 2017 roku, kiedy zaprezentowano światu tę architekturę sieci neuronowej. Kolejnym przełomem okazały się repozytoria modeli, takie jak biblioteka Transformers zespołu Hugging Face, która umożliwia łatwe pobranie wstępnie wytrenowanego modelu, jego konfigurację i użytkowanie. Poznaj niesamowite możliwości: wszędzie tam, gdzie jest mowa lub tekst, istnieją zastosowania NLP.

Tę książkę docenią praktycy: inżynierowie uczenia maszynowego i analitycy danych, poszukujący sposobu praktycznego zaadaptowania modeli transformacyjnych do swoich potrzeb. Autorzy skupili się na praktycznej stronie tworznia aplikacji językowych, a w poszczególnych rozdziałach ujęto wszystkie najważniejsze zastosowania transformerów w NLP. Zaczniesz od łatwych w użyciu potoków, następnie przystąpisz do pracy z tokenizatorami, klasami modeli i interfejsu Trainer API, pozwalającymi na trenowanie modeli do konkretnych zastosowań. Dowiesz się również, jak zastąpić interfejs Trainer biblioteką Accelerate, która zapewnia pełną kontrolę nad pętlą treningową i umożliwia trenowanie dużych transformerów od zera!

Niezwykle przejrzysty i wnikliwy przewodnik po najważniejszej bibliotece współczesnego NLP. Polecam!

Christopher Manning, profesor Uniwersytetu Stanforda

W książce:

- tworzenie modeli transformacyjnych przeznaczonych do typowych zadań NLP
- stosowanie transformerów do międzyjęzykowego uczenia transferowego
- używanie transformerów w rzeczywistych scenariuszach
- sprzęganie i wielołużywalność
- optymalizacja modeli technikami: destylacji, przycinania i kwantyzacji
- trenowanie modeli transformacyjnych z wykorzystaniem wielu procesorów graficznych w środowisku rozproszonym

Lewis Tunstall obecnie tworzy narzędziadla społeczności NLP.

Leandro von Werra zajmuje się głównie modelami generowania kodu i komunikacjąze społecznością.

Thomas Wolf jest głównym dyrektorem naukowym. Zajmuje się możliwie najszerszym upowszechnianiem wyników badań nad sztuczną inteligencją.

Helion

helion.pl

HELION SA
ul. Kościuszki 1c
44-100 Gliwice
tel.: 32 230 98 63
helion@helion.pl

KOD KORZYŚCI
Sięgnij po więcej!

ISBN 978-83-289-0711-9

9 788328 907119

Cena: 99,00 zł