**Wprowadzenie do rodziny BERT**

Ta sekcja zawiera informacje na temat modelu BERT opartego na transformatorach, a także kilku innych modeli z „rodziny BERT", a także opis niektórych różnic między tymi modelami LLM. Pierwsza część omawia różne aspekty inżynierii natychmiastowej, a także różne aspekty rozwoju LLM. Druga część wprowadza do BERT i niektórych jego funkcji, po czym dowiesz się, jak trenowany jest BERT i jakie modele BERT są dostępne. Ponadto dowiesz się, czym BERT różni się od modeli NLP sprzed wprowadzenia BERT. Trzecia część omawia BERT z punktu widzenia architektury transformatorowej, a następnie, jak wykonywać zadania NER za pomocą BERT. Ta sekcja obejmuje modelowanie tematyczne i tematyczne w BERT. Czwarta część omawia BERT i zadania czyszczenia danych, takie jak normalizacja, oraz znaczenie wyrażeń regularnych w czyszczeniu danych w BERT. Piąta część omawia warstwy osadzania BERT, a także rodzaje osadzania, takie jak osadzanie tokenów, osadzanie segmentów i osadzanie pozycji. Dowiesz się również, jak tworzyć, trenować i zapisywać model BERT. Szósta część dotyczy wewnętrznych mechanizmów BERT, takich jak MLM, NSP i specjalne tokeny BERT. Poznasz również alternatywy dla MLM (np. PLM i MLNET), a także alternatywę dla NSP (np. SOP). Ta sekcja zawiera również omówienie technik takich jak tokenizacja podsłowna, BPE, WordPiece i SentencePiece. Siódma część omawia podobieństwo zdań w BERT i kontekście słów, a następnie przykłady generowania tokenów BERT. Ósma część zawiera listę kilku modeli wytrenowanych w oparciu o BERT wraz z krótkim opisem ich funkcjonalności. Ponadto przykłady kodu wymagają obecnie języka Python 3.7, który można pobrać z Internetu. Ostatnia sekcja pokrótce przedstawia zestaw modeli opartych na BERT, z których niektóre mogą być na tyle interesujące, że warto je bardziej szczegółowo omówić za pomocą innych zasobów online.

**CZYM JEST INŻYNIERIA PODPOWIEDZI?**

Być może słyszałeś już o generatorach tekstu, takich jak GPT-3, a także o generatorach tekstu do obrazu, takich jak DALL-E 2 firmy OpenAI, Jurassic firmy AI21, Midjourney firmy Midjourney Inc. i Stable Diffusion firmy Stability AI. Inżynieria podpowiedzi odnosi się do tworzenia podpowiedzi tekstowych, które umożliwiają systemom opartym na sztucznej inteligencji generowanie wyników, które są bliższe temu, co użytkownicy chcą „pobrać" z AI. Analogicznie, podpowiedzi można porównać do roli trenerów: oferują oni porady i sugestie, aby pomóc ludziom lepiej wykonywać powierzone im zadania. Ponieważ podpowiedzi opierają się na słowach, wyzwanie polega na poznaniu, jak różne słowa mogą wpływać na generowane wyniki. Co więcej, trudno jest przewidzieć, jak systemy zareagują na daną podpowiedź. Na przykład, jeśli chcesz wygenerować krajobraz, różnica między ciemnym a jasnym krajobrazem jest intuicyjna. Jeśli jednak chcesz uzyskać piękny krajobraz, w jaki sposób system sztucznej inteligencji wygeneruje odpowiadający mu obraz? Jak można się domyślić, „konkretne" słowa są łatwiejsze do zrozumienia niż abstrakcyjne lub subiektywne dla systemów sztucznej inteligencji generujących obrazy z tekstu. Rozważmy poprzedni przykład: jak zwizualizować następujący obraz?

- Piękny krajobraz
- Piękną piosenkę
- Piękny film

Chociaż inżynieria promptów rozpoczęła się od generowania tekstu na obraz, istnieją inne rodzaje inżynierii promptów, takie jak prompty oparte na dźwięku, które interpretują podkreślony tekst i emocje wykrywane w mowie, oraz prompty oparte na szkicach, które generują obrazy z rysunków. Ostatnio w centrum uwagi znajdują się prompty oparte na tekście do generowania filmów, co stwarza ekscytujące możliwości dla artystów i projektantów.

**Monity i uzupełnienia**

Monit to ciąg tekstowy, który użytkownicy przekazują do LLM, a uzupełnienie to tekst, który użytkownicy otrzymują od LLM. Monity pomagają LLM w realizacji zlecenia (zadania) i mogą mieć różną długość. Chociaż monity mogą być dowolnym ciągiem tekstowym, w tym ciągiem losowym, jakość i struktura monitów wpływa na jakość uzupełnień. Monity można traktować jako mechanizm udzielania „wskazówek" LLM, a nawet jako sposób na „nauczenie" ich udzielania pożądanych odpowiedzi. Liczba tokenów w monicie plus liczba tokenów w uzupełnieniu może wynosić maksymalnie 2048 tokenów. Później zobaczysz przykład kodu w języku Python, który wywołuje API completion() w GPT-3.

**Rodzaje monitów**

Poniższa lista zawiera znane typy monitów dla LLM:

• monity bez powtórzeń

• monity jednorazowe

• monity z kilkoma powtórzeniami

• monity z instrukcjami

Monit bez powtórzeń zawiera opis zadania, podczas gdy monit jednorazowy składa się z pojedynczego przykładu wykonania zadania. Jak można się domyślić, monity z kilkoma powtórzeniami składają się z wielu przykładów (zwykle od 10 do 100). We wszystkich przypadkach zaleca się jasny opis zadania lub zadań: większa liczba zadań dostarcza GPT-3 więcej informacji, co z kolei może prowadzić do dokładniejszych wykonań. T0 (od „zero shot") to interesujący LLM: chociaż T0 jest 16 razy mniejszy (11 GB) niż GPT-3 (175 GB), T0 przewyższył GPT-3 w zadaniach związanych z językiem. T0 może dobrze radzić sobie z nieznanymi zadaniami NLP (tj. zadaniami, które są nowe dla T0), ponieważ został wytrenowany na zbiorze danych zawierającym wiele zadań. Poniższa strona internetowa zawiera repozytorium Github dla T0, witrynę do trenowania T0 bezpośrednio w przeglądarce, a także więcej szczegółów na temat T0 i wersji T0 o pojemności 3 GB: https://github.com/bigscience-workshop/t-zero. Jak prawdopodobnie można się domyślić, T0++ jest oparte na T0 i zostało wytrenowane za pomocą dodatkowych zadań wykraczających poza zestaw zadań, na których trenowano T0. Innym ważnym szczegółem jest to, że pierwsze trzy polecenia z powyższej listy są również nazywane odpowiednio uczeniem zero-shot, uczeniem one-shot i uczeniem few-shot.

**Polecenia instrukcyjne**

Polecenia instrukcyjne służą do precyzyjnego dostrajania LLM i określają format (określony przez użytkownika) sposobu, w jaki LLM jest komendami instrukcyjnymi. Można również uzyskać dostęp do bibliotek szablonów poleceń, które zawierają różne szablony dla różnych zadań wraz z różnymi zestawami danych.

**Odwrócone polecenia**

Inna technika wykorzystuje odwróconą kolejność: polecenia wejściowe to odpowiedzi, a odpowiedzi to pytania powiązane z odpowiedziami (podobnie jak w popularnym teleturnieju). Na przykład, mając zdanie po francusku, możesz zapytać model: „Jaki tekst angielski mógłby doprowadzić do tego francuskiego tłumaczenia?".

**Polecenia systemowe a polecenia agenta**

Rozróżnienie między poleceniem systemowym a poleceniem agenta często pojawia się w kontekście systemów sztucznej inteligencji konwersacyjnej i projektowania chatbotów. Monit systemowy to zazwyczaj początkowa wiadomość lub wskazówka dawana przez system, która wskazuje użytkownikowi, co może zrobić lub określa oczekiwania dotyczące interakcji. Często służy jako wprowadzenie lub sposób na pokierowanie użytkownikami w dalszych działaniach. Oto kilka przykładów monitów systemowych:

• „Witamy w ChatBotX! Możesz zadać mi pytania dotyczące pogody,wiadomości lub sportu. W czym mogę dziś pomóc?”

• „Dzień dobry! Aby uzyskać dane konta, naciśnij 1. Aby uzyskać pomoc techniczną, naciśnij 2.”

• „Witamy! Wpisz „zamówienie”, aby śledzić przesyłkę, lub „pomoc”, aby uzyskać pomoc.”

Z kolei monit agenta to wiadomość generowana przez model AI lub agenta w odpowiedzi na dane wprowadzone przez użytkownika w trakcie interakcji. Jest to część wymiany zdań w ramach rozmowy. Monit agenta prowadzi użytkownika do podania dodatkowych informacji, wyjaśnia niejasności lub nakłania go do podjęcia określonej czynności. Oto kilka przykładów monitów agenta: oczekuje się, że odpowiedzi będą zgodne z oczekiwaniami. Możesz przygotować własne

```
User: "I'm looking for shoes."
Agent Prompt: "Great! Are you looking for men's or
women's shoes?"
User: "I can't log in."

Agent Prompt: "I'm sorry to hear that. Can you
specify if you are having trouble with your
password or username?"
User: "Tell me a joke."
Agent Prompt: "Why did the chicken join a band?
Because it had the drumsticks!"
```

Podstawową różnicą między nimi jest ich cel i miejsce w interakcji. Monit systemowy często pojawia się na początku interakcji, przygotowując grunt pod rozmowę. Monit agenta pojawia się w trakcie rozmowy, sterując przebiegiem dialogu na podstawie danych wprowadzanych przez użytkownika. Oba typy monitów są kluczowe dla stworzenia płynnego i intuicyjnego środowiska konwersacyjnego dla użytkowników. Kierują one użytkownikiem i pomagają zapewnić, że system rozumie i skutecznie zaspokaja jego potrzeby.

**Szablony monitów**

Szablony monitów to predefiniowane formaty lub struktury używane do instruowania modelu lub systemu, aby wykonał określone zadanie. Służą one jako podstawa do generowania monitów, w których określone części szablonu można wypełnić lub dostosować, aby wygenerować różnorodne konkretne monity. Analogicznie, szablony monitów są odpowiednikiem makr, które można definiować w niektórych edytorach tekstu. Szablony monitów są szczególnie przydatne podczas pracy z modelami językowymi, ponieważ zapewniają spójny sposób przeszukiwania modelu w wielu zadaniach lub punktach danych. W szczególności szablony poleceń mogą ułatwić:

• zapewnienie spójności podczas wielokrotnego przeszukiwania modelu

• usprawnienie przetwarzania wsadowego lub automatyzacji

• zmniejszenie liczby błędów i różnic w sposobie zadawania pytań modelowi

Na przykład, załóżmy, że pracujesz z LLM i chcesz przetłumaczyć zdania z języka angielskiego na francuski. Powiązany szablon polecenia może wyglądać następująco:

„Przetłumacz następujące zdanie z języka angielskiego na francuski: {zdanie}”

Pamiętaj, że {zdanie} to symbol zastępczy, który można zastąpić dowolnym zdaniem z języka angielskiego. Możesz użyć poprzedniego szablonu polecenia, aby wygenerować konkretne polecenia:

• „Przetłumacz następujące zdanie z języka angielskiego na francuski: «Hello, how are you?»”

• „Przetłumacz poniższe zdanie z języka angielskiego na francuski: «Uwielbiam lody»”.

Jak widać, szablony podpowiedzi pozwalają łatwo generować różnorodne podpowiedzi dla różnych zdań bez konieczności przepisywania całej instrukcji za każdym razem. W rzeczywistości tę koncepcję można rozszerzyć na bardziej złożone zadania i może ona zawierać wiele symboli zastępczych lub bardziej skomplikowane struktury, w zależności od zastosowania.

**Monity dla różnych modeli LLM**

GPT-3, ChatGPT i GPT-4 to modele LLM oparte na architekturze transformatora i zasadniczo podobne pod względem mechaniki działania. ChatGPT to w zasadzie wersja modelu GPT dostosowana specjalnie do interakcji konwersacyjnych. GPT-4 to ewolucja lub ulepszenie modelu GPT-3 pod względem skali i możliwości. Różnice w monitach dla tych modeli wynikają głównie z konkretnego przypadku użycia i kontekstu, a nie z inherentnych różnic między modelami. Oto kilka różnic w monitach opartych na przypadkach użycia. GPT-3 może być używany do szerokiego zakresu zadań wykraczających poza samą konwersację, od generowania treści po pisanie kodu. Oto dwa przykłady monitów dla GPT-3:

• „Przetłumacz poniższy tekst z języka angielskiego na francuski: «Hello, how are you?»”

• „Napisz funkcję Pythona obliczającą silnię liczby”.

ChatGPT jest specjalnie dostosowana do interakcji konwersacyjnych. Oto kilka przykładów podpowiedzi dla dwóch różnych konwersacji z ChatGPT:

• Użytkownik: „Czy możesz mi pomóc z pracą domową?”

ChatGPT: „Oczywiście! Z jakim przedmiotem lub zagadnieniem potrzebujesz pomocy?”

• Użytkownik: „Opowiedz mi dowcip”.

• ChatGPT: „Dlaczego kurczak przeszedł przez plac zabaw? Żeby dostać się na drugą zjeżdżalnię!”

GPT-4 oferuje większą skalę i udoskonalenia, więc podpowiedzi będą podobne w swojej naturze do GPT-3, ale mogą dawać dokładniejsze lub bardziej zniuansowane wyniki. Oto dwa przykłady podpowiedzi dla GPT-4:

• „Przeprowadź szczegółową analizę mechaniki kwantowej w odniesieniu do ogólnej teorii względności”.

„Wygeneruj krótką historię opartą na postapokaliptycznym świecie, której tematem jest nadzieja”.

Te trzy modele akceptują podpowiedzi w języku naturalnym i generują naturalne wyniki w języku naturalnym. Podstawowy sposób interakcji z nimi pozostaje spójny. Główna różnica wynika z kontekstu, w którym używany jest model, oraz wszelkich zastosowanych dostrajań. Na przykład ChatGPT został zaprojektowany z myślą o bardziej konwersacyjnym charakterze, więc chociaż można używać GPT-3 do czatów, ChatGPT może generować bardziej istotne kontekstowo wyniki

konwersacyjne. Podczas bezpośredniej interakcji z tymi modelami, zwłaszcza za pośrednictwem API, można również kontrolować parametry takie jak „temperatura” (kontrolująca losowość) i „maksymalna liczba tokenów” (kontrolująca długość odpowiedzi). Dostosowanie tych parametrów może kształtować odpowiedzi, niezależnie od używanego wariantu GPT. Zasadniczo, chociaż modele bazowe różnią się skalą i konkretnym procesem uczenia/dostrajania, sposób ich wyświetlania pozostaje w dużej mierze spójny: jasne, konkretne podpowiedzi w języku naturalnym dają najlepsze rezultaty.

**Źle sformułowane podpowiedzi**

Tworząc podpowiedzi, staraj się być jak najbardziej jasny i konkretny, aby ukierunkować odpowiedź w pożądanym kierunku. Niejednoznaczne lub niejasne podpowiedzi mogą prowadzić do szerokiego zakresu odpowiedzi, z których wiele może nie być użytecznych lub nieistotnych dla rzeczywistych intencji użytkownika. Źle sformułowane podpowiedzi są często niejasne, wieloznaczne lub zbyt ogólne i mogą prowadzić do dezorientacji, nieporozumień lub nieprecyzyjnych odpowiedzi ze strony modeli AI. Oto kilka przykładów źle sformułowanych podpowiedzi wraz z wyjaśnieniami:

„Opowiedz mi o tym”.

Problem: Zbyt niejasne. Do jakiej „rzeczy” się odnosi?

„Dlaczego to się stało?”

Problem: Brak kontekstu. Jakie wydarzenie lub sytuacja jest omawiana?

„Wyjaśnij coś”.

Problem: Zbyt ogólne. Jakie konkretne „rzeczy” należy wyjaśnić?

„Zrób to, co konieczne”.

Problem: Niejednoznaczne. Jakie konkretne działanie jest wymagane?

„Chcę informacji”.

Problem: Niewystarczająco szczegółowe. Jakiego rodzaju informacji się oczekuje?

„Czy możesz mi przynieść tę rzecz z tego miejsca?”

Problem: Zarówno „rzecz”, jak i „miejsce” są niejasne.

„Gdzie mogę kupić książkę „jak mu tam?”

Problem: Niejednoznaczne odniesienie. Kim jest „jak mu tam”? „Jak wykonujesz ten proces?”

Problem: Do którego „procesu” się odnosi?

„Opisz wagę tematu”.

Problem: „Temat” nie jest określony.

„Dlaczego jest zły, czy dobry?”

Problem: Brak kontekstu. Co to jest „to”?

„Pomoc w rozwiązaniu problemu”.

Problem: Niejasne. Jaki konkretny problem wymaga pomocy?

„Co należy wziąć pod uwagę w tym zadaniu”.

Problem: Niejednoznaczne. Jakie „zadanie” jest omawiane?

„Jak to działa?”

Problem: Brak konkretów. Co to jest „to”? ASPEKTY ROZWOJU LLM

Jeśli przeczytałeś poprzednie sekcje, rozumiesz architekturę LLM-ów opartych wyłącznie na koderze, LLM-ów opartych wyłącznie na dekoderze oraz LLM-ów opartych zarówno na koderze, jak i dekoderze. Dla wygody w tej sekcji zamieszczono listę modeli językowych należących do każdego z tych trzech typów modeli. Mając na uwadze powyższe punkty, niektóre z bardziej znanych LLM opartych na koderze obejmują:

• AlBERT

• BERT

• DistilBERT

• ELECTRA

• RoBERTa

Poprzednie LLM dobrze nadają się do wykonywania zadań przetwarzania języka naturalnego, takich jak NER i ekstrakcyjne zadania z odpowiedziami na pytania. Oprócz LLM-ów opartych wyłącznie na koderze, istnieje kilka dobrze znanych LLM-ów opartych na dekoderze, które obejmują:

• CTRL

• GPT/GPT-2

• Transformer XK

Poprzednie modele LLM generują tekst, który może również przewidywać następne słowo. Wreszcie, niektóre z dobrze znanych modeli LLM opartych na koderze/dekoderze obejmują:

• BART

• mBART

• Marian

• T5

Poprzednie modele LLM przeprowadzają podsumowanie, tłumaczenie i generują odpowiedzi na pytania. Ostatnim trendem jest stosowanie dostrajania, treningu zero/one/few-some oraz uczenia się opartego na monitach w odniesieniu do modeli LLM. Dostrajaniu zazwyczaj towarzyszy dostrajający zbiór danych, a jeśli ten ostatni nie jest dostępny (lub niewykonalny), trening z kilkoma próbami może być akceptowalną alternatywą. Jednym z rezultatów trenowania modelu LLM Jurassic-1 jest to, że szerszy i płytszy kontekst jest lepszy niż węższy i głębszy pod względem wydajności, ponieważ szerszy kontekst pozwala na równoległe wykonywanie większej liczby obliczeń. Kolejnym wnioskiem z Chinchilli jest to, że mniejsze modele trenowane na korpusie z bardzo dużą liczbą tokenów mogą być bardziej wydajne niż większe modele trenowane na skromniejszej liczbie tokenów. Sukces GlaM i Switch LLM (oba od Google) sugeruje, że rzadkie transformatory, w połączeniu z MoE (Mixture of Experts), to również interesujący kierunek, potencjalnie prowadzący do jeszcze lepszych rezultatów w

przyszłości. Ponadto istnieje możliwość „nadmiernej kuracji” danych, co oznacza, że przeprowadzenie bardzo szczegółowej kuracji danych w celu usunięcia pozornie niepożądanych tokenów nie gwarantuje, że modele będą generować lepsze rezultaty w tych wyselekcjonowanych zbiorach danych. Zastosowanie monitów ujawniło interesujący szczegół: wyniki podobnych, ale różnych monitów mogą prowadzić do zasadniczo różnych odpowiedzi. Dlatego celem jest tworzenie dobrze skonstruowanych monitów, które są niedrogie, a jednocześnie mogą być dość trudnym zadaniem. Innym obszarem rozwoju jest ciągła potrzeba testów porównawczych wykorzystujących lepsze i bardziej złożone zbiory danych, zwłaszcza gdy modele LLM przewyższają możliwości człowieka. Konkretnie, test staje się nieaktualny, gdy wszystkie nowoczesne modele LLM przechodzą zestaw testów w tym teście. Dwa takie testy to XNLI i BigBench („Beyond the Imitation Game Benchmark”).

**Rozmiar a wydajność modelu LLM**

Rozważmy kwestię rozmiaru a wydajności: chociaż większe modele, takie jak GPT-3, mogą działać lepiej niż mniejsze, nie zawsze tak jest. W szczególności modele będące wariantami GPT-3 dają mieszane wyniki: niektóre mniejsze warianty działają prawie tak samo dobrze jak GPT-3, a niektóre większe modele działają tylko nieznacznie lepiej niż GPT-3. Ostatnim trendem jest opracowywanie modeli opartych na komponencie dekodera architektury transformatora. Takie modele są często mierzone ich wydajnością poprzez trening zero-shot, one-shot i few-shot w porównaniu z innymi modelami LLM. Ten trend, jak również rozwój coraz większych modeli LLM, prawdopodobnie utrzyma się w dającej się przewidzieć przyszłości. Co ciekawe, modele LLM oparte wyłącznie na dekoderze mogą wykonywać zadania takie jak predykcja tokenów i mogą nieznacznie przewyższać modele oparte wyłącznie na enkoderze w testach porównawczych, takich jak SuperGLUE. Jednak takie modele oparte na dekoderach są zazwyczaj znacznie większe niż modele oparte na enkoderach, a te drugie są zazwyczaj bardziej wydajne. Sprzęt to kolejny czynnik wpływający na optymalizację wydajności modelu, który może wiązać się z większymi kosztami i dlatego może być ograniczony do zaledwie kilku firm. Ze względu na wysoki koszt sprzętu, kolejna inicjatywa obejmuje szkolenie LLM na superkomputerze Jean Zay we Francji:

https://venturebeat.com/2022/01/10/inside-bigscience-the-quest-to-build-a-powerful-open-language-model/Emergent Abilities of LLMs

Emergent Abilities of LLMs odnoszą się do zdolności obecnych w większych modelach, które nie występują w mniejszych. Mówiąc w uproszczeniu, wraz ze wzrostem rozmiaru modeli następuje nieciągły „skok”, w którym zdolności ujawniają się w większym modelu bez wyraźnego ani jednoznacznego powodu. Interesującym aspektem emergentnych zdolności jest możliwość rozszerzenia możliwości modeli językowych poprzez dodatkowe skalowanie. Fizyk, laureat Nagrody Nobla, Philip Anderson, w swoim eseju z 1972 roku zatytułowanym „More Is Different” (Więcej znaczy inaczej) stwierdził:

„Emergencyja ma miejsce, gdy ilościowe zmiany w systemie powodują jakościowe zmiany w zachowaniu”.

Co ciekawe, opisano scenariusz, w którym podpowiedzi oparte na kilku próbach są uważane za emergentne (cytat z poprzedniego artykułu arXiv):

„Zdolność do wykonania zadania za pomocą podpowiedzi opartych na kilku próbach jest emergentna, gdy model ma losową wydajność do pewnego poziomu, po czym wydajność znacznie przekracza poziom losowy”. (Należy koniecznie zapoznać się z Tabelą 1 w artykule, która zawiera szczegółowe informacje dotyczące „zdolności do podpowiadania w kilku sytuacjach” (np. prawdomówność, punkt odniesienia MMLU), a także „zdolności do podpowiadania w rozszerzonym zakresie” (np. ciągłość myśli

i wykonywanie instrukcji).) Należy zauważyć, że zdolności emergentnych nie można przewidzieć poprzez ekstrapolację zachowania mniejszych modeli, ponieważ (z definicji) zdolności emergentne nie występują w mniejszych modelach. Niewątpliwie pojawią się dalsze badania badające, w jakim stopniu dalsze skalowanie modelu może prowadzić do większej liczby zdolności emergentnych w LLM.

**KAPLAN I NIEDOSZKOLONE MODELE**

Kaplan i inni przedstawili (empiryczne) prawa potęgowe dotyczące wydajności modeli językowych, które, jak twierdzą, zależą od następujących czynników:

• rozmiaru modelu

• rozmiaru zbioru danych

• ilości obliczeń do trenowania

Kaplan i inni stwierdzili, że zmiana szerokości lub głębokości sieci ma minimalny wpływ. Twierdzili również, że optymalne trenowanie bardzo dużych modeli wymaga stosunkowo niewielkiej ilości danych. Jednakże Chinchilla to model LLM 70 B, który został wytrenowany na zbiorze danych znacznie większym niż rozmiar zalecany przez Kaplana i in. W rzeczywistości Chinchilla osiągnęła status SOTA, przewyższając wydajność następujących modeli LLM, z których każdy jest od 2 do 7 razy większy od Chinchilli:

• Gopher (280 B)

• GPT-3 (175 B)

• J1-Jumbo (178 B)

• LaMDA (137 B)

• MT-NLG (530 B)

Ponadto twórcy Chinchilla LLM napisali artykuł „Scaling Laws for Neural Language Models", w którym zawarto sugerowaną liczbę tokenów dla modeli o różnych rozmiarach, które mają zostać w pełni wytrenowane, a nie niedotrenowane . Na przykład sugerowane rozmiary zestawów treningowych dla modeli o 175 B, 520 B i 1 bilionie parametrów wynoszą odpowiednio 3,7 biliona tokenów, 11,0 biliona tokenów i 21,2 biliona tokenów. Największy wpis w tej samej tabeli to modele LMM z 10 bilionami parametrów i zalecanym rozmiarem zestawu treningowego wynoszącym 216,2 biliona parametrów. Oczywiście, model LLM przekraczający 1 bilion parametrów stoi przed poważnym wyzwaniem tworzenia zbiorów danych o zalecanej wielkości, jak opisano w artykule autorów książki „Chinchilla". Jedną z interesujących możliwości jest ASR (nieomawiany w tej książce), który mógłby umożliwić generowanie zbiorów danych większych niż 10 bilionów tokenów poprzez transkrypcję dźwięku na tekst. Co więcej, niektóre spekulacje sugerują, że GPT-4 może wykorzystać ASR do utworzenia takiego zbioru danych.

**CZYM JEST BERT?**

BERT to akronim od Bidirectional Encoder Representations from transformers (Bikierunkowe Reprezentacje Kodera z Transformatorów). BERT wstępnie trenuje głębokie reprezentacje dwukierunkowe z nieoznakowanego tekstu w taki sposób, że wstępnie wytrenowany model BERT można dostroić do utworzenia modelu SOTA poprzez uwzględnienie jednej dodatkowej warstwy wyjściowej dla szerokiego zakresu zadań NLP. W rzeczywistości BERT osiągnął nowe wyniki SOTA w jedenastu zadaniach przetwarzania języka naturalnego. BERT osiągnął wyniki SOTA w zadaniach takich

jak udzielanie odpowiedzi na pytania (QnA) i wnioskowanie językowe, wprowadzając jedynie drobne zmiany w swojej architekturze. Jednocześnie BERT jest kosztowny obliczeniowo: określenie pary zdań o największym podobieństwie w korpusie 10 000 zdań wymaga ponad 60 godzin obliczeń, obejmujących około 50 milionów obliczeń wnioskowania.

**Szkolenie BERT**

BERT został wstępnie wytrenowany na dużym zbiorze nieoznakowanych danych z Wikipedii i oprócz wykonywania zadań związanych z przetwarzaniem języka naturalnego (NLP), BERT można dostroić do innych zadań. BERT wykorzystuje mechanizm uwagi do generowania osadzeń słów, które uwzględniają kontekst słów poprzedzających, jak i następujących po danym słowie. Z kolei algorytmy takie jak word2vec są bezkontekstowe, więc osadzenia słów nie zawierają żadnych informacji kontekstowych. Należy zauważyć, że etap dostrajania BERT różni się od etapu wstępnego treningu BERT: pierwszy obejmuje dane oznaczone, a drugi dane nieoznaczone.

**Modele BERT**

Dwie najbardziej znane wersje BERT to BERT-Base i BERT-Large. BERT-Base składa się z 12 warstw, 12 głowic uwagi i 110 milionów parametrów. BERT-Large to większy, wstępnie wytrenowany model, który składa się z 24 warstw, 16 głowic uwagi i 340 milionów parametrów. Istnieje jednak również kilka mniejszych modeli opartych na BERT, jak pokazano na poniższej liście:

BERT-średni L=8 i H=512

BERT-mały L=4 i H=512

BERT-mini L=4 i H=256

BERT-mały L=2 i H=128

BERT można używać w połączeniu z biblioteką transformatorów Hugging Face, która udostępnia klasy do wykonywania różnych zadań, takich jak odpowiadanie na pytania i klasyfikacja sekwencji. Ponieważ BERT nie wykorzystuje komponentu dekodera, nie nadaje się do generowania tekstu. Ponadto BERT nie nadaje się do zadań nienadzorowanych (np. klastrowania) ani do podobieństwa semantycznego.

**Architektura BERT**

BERT opiera się na architekturze transformatorowej opracowanej w 2017 roku przez Google. BERT wymaga jedynie części kodującej architektury transformatorowej do wygenerowania modelu reprezentacji języka. Słownik BERT obejmuje około 30 000 słów, a każde osadzenie słowa w BERT ma 768 cech, co oznacza, że każde osadzenie słowa jest wektorem liczb zmiennoprzecinkowych 1x768. Model BERT składa się z części kodującej architektury transformatorowej, co oznacza, że jest to architektura wyłącznie kodująca. Główną częścią BERT jest bardzo duża, 12-warstwowa sieć neuronowa przetwarzająca tekst. Można jednak zastąpić ostateczny klasyfikator w BERT modelem specyficznym dla danego zadania, co pozwala wykorzystać całą wiedzę BERT. Ten końcowy etap szkolenia nazywany jest „dostrajaniem", ponieważ ilość czasu potrzebnego na dostosowanie BERT do własnego zadania jest bardzo mała w porównaniu z tym, co Google potrzebowało na wstępne wyszkolenie BERT. Etap dostrajania jest proporcjonalnie mniejszym zadaniem, które nadal może wymagać znacznej ilości obliczeń.

**Funkcje BERT**

BERT posiada zestaw około 30 000 zapamiętanych wektorów surowych. Co więcej, około 80% tych wektorów surowych odpowiada „normalnym" słowom (tj. występującym w słowniku języka

angielskiego). Pozostałe 20% to podsłowa tworzone przez WordPiece: te podsłowa mają formę „##s” lub „##ed”. Te ostatnie są przydatne do wykrywania czasu przeszłego czasownika w zdaniu. Ponadto słownik BERT składa się w 45% z wielkich i w 25% z małych liter (w przybliżeniu).

**Ograniczenia BERT**

Chociaż BERT może wykonywać zadania związane z tekstem, interpretować tekst i tworzyć przewidywania, BERT nie może wykonywać następujących zadań:

• generować nowego tekstu

• tłumaczyć tekstu między językami

• generować odpowiedzi na pytania

Przyczyną powyższych ograniczeń jest to, że BERT opiera się wyłącznie na komponencie kodującym architektury transformatora. Natomiast modele LLM oparte na komponencie dekodującym, takie jak modele oparte na GPT, są w stanie wykonywać wszystkie zadania z powyższej listy. Jednak oryginalny model transformatora został wytrenowany na ręcznie tworzonych przykładach tłumaczeń, podczas gdy BERT to wstępnie wytrenowany model, który wykorzystywał znacznie większy zbiór danych składający się z tekstu nieoznaczonego. Jak trenowany jest BERT? BERT trenuje się, wykonując najpierw krok wstępnego trenowania oparty na danych specyficznych dla danego zadania, a następnie krok precyzyjnego dostrajania. Na przykład, aby przeprowadzić analizę sentymentu za pomocą BERT, potrzebny jest korpus oznaczonych danych, który określa, czy zdanie ma pozytywny, czy negatywny sentyment. Ponadto zbiór danych jest dzielony na część treningową i część testową, tak jak widzieliśmy w przykładach kodu w poprzednich rozdziałach. Jest to również typowe dla algorytmów uczenia maszynowego, które obejmują uczenie nadzorowane, takich jak regresja liniowa i algorytmy klasyfikacji. Przed etapem treningu BERT przeprowadza analizę wielopoziomową (MLM), która obejmuje 1) zastąpienie około 15% tokenów tokenem [MASK] oraz 2) zastąpienie dodatkowego, losowo wybranego zestawu słów innym (tj. niepoprawnym) zestawem. Rzeczywisty etap szkolenia obejmuje trenowanie BERT na losowo wybranych sekwencjach 512 tokenów, a celem jest, aby BERT dokonywał predykcji dla wszystkich tokenów wejściowych. Oprócz MLM, BERT wykonuje predykcję następnego zdania (NSP). Zadanie to polega na łączeniu par zdań: niektóre pary to zdania sąsiadujące, a inne składają się ze zdań z losowych lokalizacji. Celem tego zadania jest, aby BERT poprawnie przewidywał, czy pary zdań są ciągłe. Po zakończeniu etapu wstępnego szkolenia, etap dostrajania obejmuje trenowanie modelu na zestawie przykładowych zadań. Na przykład, aby wytrenować BERT do wykonywania zadania polegającego na odpowiadaniu na pytania, należy dostroić wstępnie wytrenowany model (który został wytrenowany na powiązanym zadaniu) na korpusie danych pytań i odpowiedzi.

**BERT-Base i BERT-Large**

BERT-Base składa się z 12 warstw kodera, ułożonych jedna na drugiej. Wszystkie kodery wykorzystują 12 głowic uwagi. Sieć sprzężenia zwrotnego w koderze składa się z 768 ukrytych jednostek, więc rozmiar reprezentacji uzyskanej z bazy BERT również wyniesie 768. Dla porównania, model BERT-Large składa się z 24 pionowo ułożonych warstw kodera, z których każda ma 16 głowic uwagi. Sieć sprzężenia zwrotnego w każdej warstwie kodera składa się z 1024 ukrytych jednostek. Zatem rozmiar reprezentacji uzyskanej z modelu BERT-Large wyniesie 1024. Ponadto model BERT-Large zawiera 340 milionów parametrów.

**BERT a wcześniejsze modele przetwarzania języka naturalnego (NLP)**

Istnieje kilka ważnych aspektów BERT, które odróżniają BERT od modeli opartych na algorytmach (takich jak word2vec). Po pierwsze, BERT nie wykonuje operacji stemmingu: zamiast tego BERT przeprowadza tokenizację podwyrazów za pomocą programu WordPiece. Główna różnica polega na tym, że stemming odrzuca sufiksy, podczas gdy WordPiece ich nie odrzuca. Po drugie, BERT tworzy kontekstowe osadzenia słów, podczas gdy word2vec tworzy dystrybucyjne osadzenia słów. Mówiąc dokładniej, BERT wykorzystuje wszystkie słowa w zdaniu do wygenerowania osadzenia słów dla każdego słowa w danym zdaniu, a każde osadzenia słów są specyficzne dla zdania, w którym słowo występuje. W rezultacie słowo występujące w wielu zdaniach ma różne osadzenia słów, podczas gdy word2vec używa bigramów do obliczenia pojedynczego osadzenia słów w zdaniach. Po trzecie, BERT nie wykorzystuje podobieństwa kosinusowego (to znaczy nie bezpośrednio) do określania stopnia podobieństwa dwóch słów. Możliwe jest jednak użycie BERT z podobieństwem kosinusowym, pod warunkiem dostrojenia BERT do odpowiednich danych, takich jak dane i przykłady kodu w następującym repozytorium:

https://github.com/UKPLab/sentence-transformers

**Samouwaga w BERT**

W sekcji 1 omówiliśmy mechanizm uwagi w modelach transformatorowych. Różne typy modeli LLM wykorzystują różne typy samouwagi. W szczególności modele oparte na koderze, takie jak BERT, wykorzystują dwukierunkową samouwagę. Modele LLM oparte na dekoderze, takie jak GPT, wykorzystują jednostronną samouwagę, co oznacza, że każdy token uwzględnia tylko kontekst po swojej lewej stronie (ale nie po prawej).

**Jak BERT przetwarza osadzenia wejściowe**

W przeciwieństwie do sieci neuronowych RNN i LSTM, które przetwarzają tokeny słów w sposób sekwencyjny, BERT przetwarza zestaw osadzeń wejściowych równolegle, generując osadzenia składające się z 768 liczb zmiennoprzecinkowych dla każdego osadzenia wejściowego. BERT może również przetwarzać zdania o zmiennej długości, ponieważ generuje pojedynczy wektor ważony poprzez głębokie transformacje dwukierunkowe, które uwzględniają relacje kontekstowe między słowami w zdaniu, a nie po prostu uśredniają osadzenia słów. Należy zauważyć, że chociaż architektura transformatora umożliwia równoległe przetwarzanie słów w zdaniu, jest to cecha mechanizmu samouwagi, a nie równoległe obliczanie średnich ważonych. Reprezentacją wejściową dla BERT jest pojedyncze zdanie tekstowe lub para zdań tekstowych. W przypadku dwóch zdań, każdy token w pierwszym zdaniu otrzymuje osadzenie A, a każdy token w drugim zdaniu otrzymuje osadzenie B, a zdania są rozdzielone tokenem [SEP]. Sekwencje wejściowe dla BERT mogą mieć maksymalną długość 512 tokenów: inne modele obsługują maksymalną długość sekwencji wejściowej wynoszącą 1024 tokeny. Domyślnie BERT obsługuje dwa zadania wstępnego trenowania: MLM i NSP, które zostały pokrótce omówione wcześniej w tej sekcji i zostaną omówione również w dalszej części. Pyramid-BERT Amazon stworzył Pyramid-BERT, technikę redukującą różne aspekty modeli opartych na BERT, z jedynie niewielką redukcją dokładności o 1,5%:

• skrócony czas trenowania

• skrócony czas wnioskowania

• zmniejszone zużycie pamięci

Oprócz powyższych ulepszeń, modele BERT mogą przetwarzać dłuższe sekwencje tekstowe. Pyramid-BERT osiąga swoje udoskonalenie poprzez przetwarzanie warstw pośrednich w celu usunięcia

niektórych osadzeń pojedynczych słów. W szczególności Pyramid-BERT usuwa redundancję istniejącą w tokenie [CLS] po przetworzeniu tokenów przez końcowy koder.

**BERTScore**

BERTScore porównuje znaczenie wygenerowanego zdania z tym, co miało zostać wygenerowane. Biorąc pod uwagę zdanie i zdanie wygenerowane, BERTScore jest obliczany w następujący sposób:

• Przepuść oba zdania przez wstępnie wytrenowany model BERT.

• Wygeneruj osadzenie kontekstowe dla każdego słowa.

• Wykonaj obliczenia n-kwadrat.

• Oblicz podobieństwo każdego słowa odniesienia do każdego słowa kandydującego.

• Wybierz słowo odniesienia, które jest najbardziej podobne do słowa kandydującego.

• Oblicz precyzję, wskaźnik przypominania i wynik f.

**ZADANIA BERT I NLP**

Modele NLP mogą wykonywać różnorodne zadania o różnym stopniu dokładności, z których niektóre wymieniono poniżej:

• Klasyfikacja dokumentów

• Pytania i odpowiedzi

• Tekst specyficzny dla danej dziedziny

• Aplikacje w językach innych niż angielski

• Rozpoznawanie jednostek nazwanych

• Łączenie tekstu z innymi typami obiektów

Należy pamiętać, że klasyfikacja dokumentów może obejmować klasyfikację wieloklasową i klasyfikację wieloetykietową (tj. punkt danych ma więcej niż jedną etykietę). BERT odniósł sukces w wielu zadaniach, takich jak rekomendacje oparte na treści, udzielanie odpowiedzi na pytania (QnA) i przetwarzanie wideo. W fazie wstępnego treningu bez nadzoru BERT może przetwarzać bardzo duże, nieoznakowane zbiory danych, po czym następuje faza dostrajania, która obejmuje jedynie ograniczoną ilość danych oznaczonych. W związku z tym modele językowe oparte na BERT są wysoce pożądane, ponieważ osiągają dokładność SOTA. Ponieważ przetwarzanie niezwykle dużego korpusu jest czasochłonne, skrócenie czasu uczenia jest możliwe dzięki technice znanej jako „równoległe trenowanie modelu danych”, która polega na podziale zbioru danych treningowych na wiele podzbiorów, z których każdy jest przetwarzany za pomocą repliki modelu. Co więcej, obliczone gradienty są redukowane do średniej we wszystkich replikach modelu podczas każdego kroku optymalizacji, co zapewnia identyczność aktualizacji wag oraz stanu modelu we wszystkich replikach.

**Słownictwo BERT i słowa poza słownikiem**

Oryginalny model BERT ma stały rozmiar słownika wynoszący łącznie około 30 000 tokenów. Od trzech piątych do czterech piątych z nich to w rzeczywistości całe słowa, a pozostały zestaw słów składa się z podsłów lub „fragmentów słów”. BERT reprezentuje dany token wejściowy za pomocą kombinacji

osadzeń, które wskazują odpowiadający mu token, segment i pozycję. Gdy BERT natrafi na słowo spoza słownika (OOV), jego tokenizator używa modułu WordPiece do rozbicia słowa na zestaw podsłów.

**BERT i kolejność wyrazów**

BERT nie przechowuje informacji o kolejności wyrazów: można go zastosować do słów w dowolnej kolejności (lub wszystkich naraz równolegle, jak to robimy w praktyce). Natomiast sieci neuronowe RNN i LSTM mogą przechowywać kolejność wyrazów, ponieważ słowa są wprowadzane sekwencyjnie. Dane słowo jest przetwarzane poprzez połączenie bieżącego słowa z wynikiem uzyskanym z poprzednio przetworzonego słowa. BERT uwzględnia kolejność wyrazów poprzez zastosowanie osadzania pozycyjnego, które ma na celu dostarczenie modelowi informacji o pozycji każdego słowa w sekwencji. BERT ma odrębne osadzanie pozycyjne dla każdej możliwej pozycji w sekwencji aż do jej maksymalnej długości (512 dla BERT).

**BERT i zdania „zbyt długie"**

BERT-Base może przetworzyć maksymalnie 512 tokenów wejściowych, ponieważ jest to maksymalna długość narzucona przez PE długości sekwencji wejściowej. Jeśli zdania wejściowe przekraczają tę maksymalną liczbę, dostępne są następujące opcje:

1. Użyć większego modelu BERT, który obsługuje sekwencje wejściowe o długości 2048 tokenów.

2. Skrócić zdania wejściowe dłuższe niż 512 tokenów.

3. Odrzucić zdania, których długość przekracza 512 tokenów (niezalecane).

4. Podzielić każde zdanie na fragmenty po 512 tokenów.

Opcja nr 3 jest najmniej zalecanym podejściem: jest analogiczna do radzenia sobie z wartościami odstającymi poprzez ich odrzucanie. Chociaż opcja nr 4 jest atrakcyjna, mogą istnieć logicznie powiązane zależności między fragmentami, które nie zostałyby uwzględnione przez tę technikę.

**Dopełnianie zdań**

BERT można zastosować do zdań o dowolnej długości do 512 tokenów. Jednak, aby obsługiwać równoległe przetwarzanie wielu próbek tekstu jednocześnie, implementacje BERT wymagają określenia jednej stałej długości dla całego tekstu wejściowego. W celu zapewnienia paralelizacji wszystkie sekwencje tekstu wejściowego są uzupełniane lub skracane do pojedynczej stałej długości, a

• krótsza stała długość przyspieszy trenowanie i ewaluację BERT.

• BERT ma sztywny limit 512 tokenów.

**Uzupełnienie a obcięcie**

Uzupełnienie jest wymagane, gdy zdania wejściowe lub dokumenty mają różną długość. Znak uzupełnienia może mieć wartość 0 lub inną wartość specyficzną dla algorytmu. Uzupełnienie z lewej strony poprzedza zdanie lub dokument znakami uzupełnienia, natomiast uzupełnienie po nim dodaje znaki uzupełnienia do zdania lub dokumentu. Ponadto istnieje kilka sposobów określenia liczby wymaganych znaków uzupełnienia, z których niektóre wymieniono tutaj:

• Dynamiczne uzupełnianie dokumentów dla każdej partii.

• Uzupełnianie każdego dokumentu do długości najdłuższego dokumentu.

• Uzupełnianie każdego dokumentu do średniej długości dokumentów.

Pierwsza opcja polega na dynamicznym uzupełnianiu dokumentów, tak aby długość uzupełnienia była oparta na bieżącej partii dokumentów. Uzupełnianie oparte na najdłuższym dokumencie jest niezalecane, gdy występują wartości odstające od długości dokumentów. Na przykład, jeden dokument może zawierać 50 tys. tokenów, podczas gdy każdy inny dokument zawiera maksymalnie 1 tys. tokenów. Obcinanie polega na określeniu maksymalnej długości dokumentu, a następnie po prostu odrzuceniu tokenów, których pozycje w dokumentach przekraczają określoną długość maksymalną. Niestety, ta technika może prowadzić do utraty cennych informacji zawartych w odrzuconych tokenach. Zmodyfikowana forma obcinania polega na dopełnieniu krótszych dokumentów do określonej długości (np. 1 tys.), a następnie podzieleniu dokumentu 50 tys. tokenów na zestaw 50 krótszych dokumentów zawierających 1 tys. tokenów. Jednak podzielenie dokumentu w ten sposób może spowodować utratę informacji kontekstowych w pierwszych kilku tokenach, do których odwołuje się pod koniec zestawu dokumentów częściowych. Jeśli użyjesz średniej długości dokumentu jako wartości maksymalnej długości dokumentu, możesz również napotkać utratę informacji z powodu odrzucenia tokenów. Kolejny ważny szczegół: obcinanie dokumentów nie wpływa na zadania takie jak analiza sentymentu i wykrywanie spamu, ponieważ ich prognozy składają się z pojedynczej wartości. Jednak obcinanie dokumentów może negatywnie wpłynąć na tłumaczenie maszynowe i NER. BERT ma limit 512 tokenów wejściowych, natomiast GPT-2 ma limit 1024 tokenów wejściowych, co oznacza, że należy przestrzegać tych ograniczeń (oraz ograniczeń innych modeli języka) pod względem długości dokumentów wejściowych.

## BERT I ARCHITEKTURA TRANSFORMATOR

BERT to jeden z dobrze znanych, wczesnych, wstępnie wytrenowanych modeli językowych opartych na architekturze Transformera, co zostało nazwane „momentem ImageNet dla przetwarzania języka naturalnego". Architektura BERT składa się z komponentu wyłącznie kodera, który zawiera wiele pionowo ułożonych warstw kodera. BERT został wytrenowany w sposób samonadzorowany na ogromnych zbiorach danych. BERT uzyskał statystyczne zrozumienie języka i mógł funkcjonować jako model językowy. Istnieje wiele wariantów wstępnie wytrenowanych modeli BERT, które są wstępnie trenowane na różnych typach zbiorów danych (takich jak prawne, biomedyczne, naukowe itd.). Wiele z tych wariantów modeli charakteryzuje się ulepszonymi parametrami wydajnościowymi (takich jak RoBERTa, ALBERT i DistilBERT) i zostały one wstępnie wytrenowane przez duże organizacje, które następnie udostępniły te modele do powszechnego użytku. Ponieważ wstępne trenowanie jest procesem kosztownym i czasochłonnym, dostępność mnóstwa wstępnie wytrenowanych modeli stanowi ogromną korzyść dla społeczności. Co ciekawe, pierwszy model GPT z OpenAI (omówiony w rozdziale 7) poprzedzał BERT i również miał wpływ na jego rozwój. OpenAI wykorzystywał modelowanie języka, czyli zadanie uczenia się bez nadzoru, które przewiduje kolejne słowo w danym zdaniu.

### BERT i dwukierunkowość

Jak sama nazwa wskazuje, BERT obsługuje dwukierunkowość w przetwarzaniu sekwencji danych wejściowych. W szczególności BERT przetwarza sekwencje danych wejściowych jako całe zdania, a nie pojedyncze tokeny, aby uzyskać kontekst zarówno z kierunku od lewej do prawej, jak i od prawej do lewej tekstu wejściowego. Inne modele, takie jak ELMo i GPT, wykorzystują technikę jednokierunkową. ELMo faktycznie obsługuje mechanizm dwuprzebiegowy: model jest trenowany od lewej do prawej, a drugi model jest trenowany od prawej do lewej, a następnie oba modele są łączone w celu utworzenia jednego modelu. Niemniej jednak dwukierunkowość BERT przewyższa dwufazowe podejście ELMo.

## BERT I PRZETWARZANIE TEKSTU

W tej sekcji i kilku podsekcjach omówiono, jak BERT wykonuje następujące zadania NLP:

• Klasyfikacja tokenów

• Rozpoznawanie jednostek nazwanych (NER)

• Klasyfikacja tekstu

Klasyfikacja tokenów

Zadania na poziomie tokenów obejmują predykcję dla każdego tokena w danych wejściowych. BERT pobiera zestaw osadzeń słów i generuje zestaw lepszych osadzeń słów, dlatego zadania na poziomie tokenów są najbardziej „naturalnym" zastosowaniem.

**BERT i NER**

Modele językowe oparte na BERT można dostroić do zadań dalszych, takich jak NER i RE, dla wyspecjalizowanych domen. Przypomnijmy , że każdy token w sekwencji wejściowej jest kombinacją trzech wektorów: osadzenia słowa, osadzenia segmentu i osadzenia pozycji. Możesz użyć BERT do wykonania NER, dodając gęstą warstwę klasyfikacji na wierzchu modelu BERT. Rozmiar tej warstwy odpowiada liczbie nazwanych kategorii encji, które chcesz rozpoznać (np. OSOBA, ORGANIZACJA i LOKALIZACJA, plus etykieta „O" dla tokenów, które nie są częścią żadnej nazwanej encji). Teraz możesz dostroić model w zbiorze danych NER. Dla każdego tokena w danych wejściowych należy użyć tokenizatora BERT do wygenerowania kontekstualizowanego osadzenia. To osadzenie jest przepuszczane przez warstwę klasyfikacji w celu wygenerowania predykcji dla tego tokena. Model jest następnie trenowany w celu zminimalizowania różnicy między jego predykcjami a rzeczywistymi etykietami NER. Być może zastanawiasz się, jak przetwarzane są predykcje dla podsłów: w końcu mogą to być fragmenty słów, które same w sobie nie mają żadnego znaczenia. W rezultacie wymagane są pewne kroki post-processingu, aby połączyć predykcje dla podsłów należących do tego samego słowa i zapewnić spójne rozpoznawanie encji.

**Klasyfikacja tekstu**

BERT można zastosować w tradycyjnych zadaniach przetwarzania języka naturalnego (NLP), takich jak klasyfikacja tekstu. Ponieważ BERT generuje zbiór osadzeń, możemy wykonać pewne przetwarzanie (takie jak uśrednianie ich wartości), a następnie przekazać wynik do klasyfikatora. Następnie na model BERT dodawana jest gęsta warstwa klasyfikacji. Rozmiar tej warstwy odpowiada liczbie klas w zadaniu klasyfikacji. Tylko reprezentacja tokena [CLS] (z ostatniej warstwy BERT) jest używana jako zagregowana reprezentacja sekwencji i przekazywana do warstwy klasyfikacji. Innymi słowy, BERT przetwarza całą sekwencję, a następnie wykorzystuje kontekstualizowaną reprezentację tokena [CLS] do przeprowadzenia klasyfikacji.

**ZADANIA BERT I CZYSZCZENIA DANYCH**

Czyszczenie danych to podstawowy proces uczenia maszynowego, którego kroki zależą od rodzaju danych i modelu. Na przykład algorytmy klasyfikacyjne, takie jak kNN i SVM, obejmują obsługę brakujących i niepoprawnych danych jako zadania czyszczenia danych, podczas gdy zbiory danych związane z przetwarzaniem języka naturalnego (NLP) mogą obejmować normalizację. Poniższa lista zawiera zadania czyszczenia danych, które są istotne dla zbiorów danych dla modeli opartych na BERT:

• Usuwanie słów pomijanych.

• Przeprowadzanie normalizacji.

• Używanie wyrażeń regularnych.

• Brak stemmingu.

• Przeprowadzanie lematyzacji.

Pierwsze trzy punkty są jednak zadaniami „warunkowymi” z następujących powodów.

1. Usuwanie słów pomijanych może czasami pomóc w redukcji szumu i skupieniu się na najbardziej informacyjnych słowach w tekście. Jednak BERT został zaprojektowany do pracy z surowymi danymi tekstowymi, a usuwanie słów pomijanych może czasami spowodować utratę ważnych informacji kontekstowych, które BERT mógłby wykorzystać.

2. Normalizacja może obejmować zadania takie jak konwersja całego tekstu na małe litery lub poprawianie literówek, co może być korzystne w standaryzacji danych tekstowych. Jednak BERT jest wstępnie trenowany na dużych korpusach bez rozległej normalizacji, aby zachować niuanse języka naturalnego, więc rozległa normalizacja nie zawsze jest konieczna.

3. Użycie wyrażeń regularnych może być przydatne do czyszczenia i strukturyzacji danych tekstowych, na przykład usuwania niepożądanych znaków lub wzorców. Należy jednak używać ich rozważnie, aby uniknąć usuwania ważnych informacji. Po oczyszczeniu zbioru danych, kolejnym krokiem jest podział zbioru danych na podzbiory do trenowania i testowania, zazwyczaj za pomocą funkcji train_test_split() biblioteki Scikit-learn.

**Normalizacja**

Normalizacja to proces skalowania kolumn liczbowych w zbiorze danych, tak aby miały wspólną skalę. Dodatkowo, skalowanie odbywa się w następujący sposób:

1. skalowanie wartości do zakresu [0,1]

2. bez utraty informacji

3. bez zniekształcania różnic występujących w zakresach wartości.

Normalizację danych można przeprowadzić za pomocą klasy MinMaxScaler w bibliotece scikit-learn. Po oczyszczeniu zbioru danych należy wykonać inne zadania, takie jak podział zbioru danych na podzbiory do trenowania i testowania, zazwyczaj za pomocą funkcji train_test_split() biblioteki scikit-learn. Następnie zazwyczaj przeprowadza się tokenizację, która zostanie omówiona w dalszej części.

**Wyrażenia regularne**

Wyrażenia regularne zapewniają zaawansowane funkcje wyszukiwania i zastępowania tekstu w ciągu tekstowym, dokumencie lub korpusie. Ponieważ książka koncentruje się na architekturach i modelach opartych na transformatorach, pominięto przykłady kodu dla wyrażeń regularnych. Jeśli chcesz oswoić się z wyrażeniami regularnymi lub nabrać w nich biegłości, dostępne są bezpłatne zasoby online.

**TRZY WARSTWY OSADZANIA BERTA**

BERT ma trzy typy warstw osadzania:

• osadzanie tokenów

• osadzanie segmentów

• osadzanie pozycji

Osadzenie słów w BERT składa się z jednowymiarowego wektora z 768 liczbami zmiennoprzecinkowymi dla każdego tokena, które są tworzone poprzez sumę składowych trzech wektorów zmiennoprzecinkowych, jak opisano w następnym akapicie. Każde osadzenie tokena ma kształt (1, 768), który jest po prostu wektorową reprezentacją słowa. Każde osadzenie segmentu ma kształt (1, 768), który jest wektorową reprezentacją ułatwiającą BERT rozróżnianie sparowanych sekwencji wejściowych. Każde osadzenie pozycji ma kształt (1, 768), który umożliwia BERT śledzenie pozycji tokenów w sekwencji wejściowej. Trzy poprzednie wektory są dodawane, tworząc kolejny wektor, który jest przekazywany do warstwy kodującej dla BERT.

**Trzy typy osadzeń pozycyjnych**

BERT może przetwarzać sekwencje wejściowe o maksymalnej długości 512. Ponieważ BERT działa z sekwencjami o stałej długości, należy określić maksymalną długość zestawu sekwencji wejściowych o zmiennej długości, które muszą zostać skrócone do 512. Zdania zawierające mniej niż 512 tokenów można uzupełnić, aby miały długość 512. Następnym krokiem jest obliczenie zgodnego wektora zawierającego informacje o położeniu dla każdego tokena. W szczególności warstwę osadzania można przedstawić za pomocą tabeli (512,768), tak aby i-ty wiersz zawierał reprezentację wektorową słów na i-tej pozycji. BERT posiada również specjalny token, [CLS], dla pierwszej pozycji i token [SEP] dla separatora między kolejnymi zdaniami. Wygenerowany wynik składa się z sumy składowych trzech wektorów: wektora z osadzonymi tokenami, wektora z osadzonymi segmentami i osadzonych pozycji wektora. Przypomnijmy, że uwaga przyczynowa oznacza, że tokeny mogą uzyskać dostęp tylko do poprzednich tokenów, które są częścią tego samego ciągu tekstowego, podczas gdy uwaga nieprzyczynowa nie ma tego ograniczenia. Ponadto istnieją następujące typy technik kodowania związanych z pozycją:

• osadzanie pozycji

• kodowanie pozycji

• pozycje względne

Główne punkty dotyczące BERT obejmują następujące szczegóły architektoniczne i związane z trenowaniem:

• uwaga nieprzyczynowa (w porównaniu z GPT-2)

• osadzanie pozycji

• nienadzorowane wstępne trenowanie

• nadzorowane dostrajanie

**TWORZENIE MODELU BERT**

Pakiet transformatorów Hugging Face zawiera klasę AutoModel, która umożliwia utworzenie instancji modelu z punktu kontrolnego. Ta klasa jest klasą opakowującą, która została zaprojektowana w taki sposób, aby można było określić poprawny model dla danego punktu kontrolnego. Alternatywnie, można określić jawnie klasę, jeśli posiada się zaawansowaną wiedzę na temat wymaganego modelu. Przyjrzyjmy się, jak to działa w przypadku modelu BERT. Na listingu zaprezentowano zawartość pliku bert_model.py, która ilustruje sposób tworzenia modelu BERT.

```
from transformers import BertConfig, BertModel

# 1) initialize the config object
config = BertConfig()

# 2) initialize a BERT model:
model = BertModel(config)

# display the contents of the configuration object:
print(config)
```

Listing  rozpoczyna się od polecenia import, a następnie inicjuje zmienną config jako instancję klasy BertModel. Następnie zmienna model jest inicjowana jako instancja BertModel wraz ze zmienną config. Ostatni fragment kodu wyświetla zawartość zmiennej config. Uruchom kod z Listingu , a zobaczysz następujący wynik:

```
BertConfig {
  "attention_probs_dropout_prob": 0.1,
  "classifier_dropout": null,
  "gradient_checkpointing": false,
  "hidden_act": "gelu",
  "hidden_dropout_prob": 0.1,
  "hidden_size": 768,
  "initializer_range": 0.02,
  "intermediate_size": 3072,
  "layer_norm_eps": 1e-12,
  "max_position_embeddings": 512,
  "model_type": "bert",
  "num_attention_heads": 12,
  "num_hidden_layers": 12,
  "pad_token_id": 0,
  "position_embedding_type": "absolute",
  "transformers_version": "4.10.3",
  "type_vocab_size": 2,
  "use_cache": true,
  "vocab_size": 30522
}
```

**TRENOWANIE I ZAPISYWANIE MODELU BERT**

W poprzedniej sekcji pokazano, jak utworzyć model BERT, a ta sekcja pokazuje, jak go zapisać. Listing 2 przedstawia sposób tworzenia modelu BERT i zapisywania go na komputerze.

```
from transformers import BertConfig, BertModel
from transformers import BertConfig, BertModel

# initialize the config object
config = BertConfig()

# BERT model initialized with random values:
model = BertModel(config)

#model.save_pretrained(a_directory_location)
model.save_pretrained("/tmp")
```

Listing 2 rozpoczyna się od zaimportowania kilku klas z biblioteki transformatorów, a następnie inicjalizacji zmiennych config i model, tak jak w Listingu 1. Ostatni fragment kodu zapisuje odpowiedni model w katalogu /tmp, który można zapisać w innej lokalizacji. Uruchom kod z Listingu 2, a w katalogu /tmp zobaczysz następujący wynik:

```
-rw-r--r-- 1 618 Nov 4 13:15 config.json
-rw-r--r-- 1 438013119 Nov 4 13:15 pytorch_model.
bin
```

Plik pytorch_model.bin nazywany jest „słownikiem stanów” i jest dużym plikiem, ponieważ zawiera wszystkie wagi modelu. Plik config.json zawiera atrybuty architektury modelu.

**WEWNĘTRZNE DZIAŁANIE BERTA**

BERT implementuje szereg interesujących technik, z których niektóre wymieniono tutaj:

• MLM (model języka maskowanego)

• NSP (prognozowanie następnego zdania)

• Tokeny specjalne ([CLS] i [SEP])

• Maska językowa

• Wordpiece (tokenizacja podsłów jest używana w BERT)

• SentencePiece (używana w ALBERT)

Każdy temat z powyższej listy jest krótko omówiony w kolejnych podsekcjach, wraz z PLM i MLNET (oba są alternatywami dla MLM) oraz SOP (alternatywą dla NSP).

**Czym jest MLM?**

Model języka maskowanego (MLM) to nienadzorowane zadanie wstępnego treningu BERT na nieoznakowanych danych, podczas którego BERT przetwarzał zawartość Wikipedii oraz BookCorpus. W tym zadaniu około 15% słów zostało zastąpionych tokenem [MASK], a następnie BERT przewidywał brakujące słowa. Należy zauważyć, że zadanie to zostało wykonane na „fragmentach” danych przesłanych do BERT. Kolejnym zadaniem wstępnego treningu jest przewidywanie następnego zdania (NSP). Wiele słów w Wikipedii zawiera daty, imiona i nazwiska osób oraz nazwy miejscowości, z których niektóre zostały zastąpione tokenem [MASK]. Podczas procesu szkolenia BERT poprawnie zidentyfikował brakujące tokeny. Warto zauważyć, że dodatek zawiera informacje dotyczące transferu wiedzy, wstępnego szkolenia i dostrajania modeli językowych.

**PLM i MPNet: Alternatywy dla MLM**

W poprzedniej sekcji poznałeś/aś MLM, technikę polegającą na wyborze 15% tokenów z korpusu i zastąpieniu około 80% z nich tokenem [MASK]. Modele takie jak BERT próbują znaleźć prawidłowe tokeny podczas etapu wstępnego szkolenia. Inną techniką jest permutowane modelowanie języka (PLM), które bada zależności między przewidywanymi tokenami. W przeciwieństwie do BERT, który nie wykonuje PLM, XLNet obsługuje PLM w celu określenia tych zależności. Chociaż XLNet wprowadza operację permutacji, która może potencjalnie zmieniać pierwotne pozycje tokenów, nadal zachowuje informacje o położeniu dzięki architekturze transformatora, która z natury uwzględnia kodowanie pozycyjne. Operacja permutacji może jednak utrudniać uczenie się rzeczywistych relacji pozycyjnych między tokenami. Trzecią techniką jest MPNet, metoda wstępnego trenowania, która zapewnia zalety BERT i XLNet, jednocześnie unikając ich ograniczeń. Kod i wstępnie wytrenowane modele wykorzystujące MPNet są dostępne na stronie https://github.com/microsoft/MPNet.

**Czym jest NSP?**

Przewidywanie następnego zdania (NSP) to gotowe zadanie wstępnego treningu wykonywane przez BERT (drugim zadaniem wstępnego treningu jest MLM). NSP łączy pary zdań w następujący sposób:

• Drugie zdanie jest logicznie powiązane z pierwszym zdaniem w 50% par.

• Drugie zdanie nie jest logicznie powiązane z pierwszym zdaniem w 50% par.

Celem NSP jest identyfikacja par zdań poprawnych i niepoprawnych. Należy wspomnieć o jeszcze jednej kwestii. NSP jest zazwyczaj opisywane w kategoriach „par zdań", co można doprecyzować w następujący sposób:

• „Zdanie" to „zakres tekstu".

• „Zakres tekstu" może być dłuższy niż zdanie.

• „Zakres tekstu" może być krótszy niż zdanie.

• Długość pary musi być mniejsza niż 512 tokenów.

**NSP kontra SOP**

Chociaż NSP jest wartościowym zadaniem do wstępnego trenowania, istnieją alternatywy. Przewidywanie kolejności zdań (SOP) to przykład, który w pewnym stopniu przypomina NSP. Ciekawostką dotyczącą SOP jest to, że czasami dwa fragmenty tekstu są prezentowane w odwrotnej kolejności, a nie w „normalnej" kolejności. W rzeczywistości architekci RoBERTa i XLNet zastąpili NSP SOP, ponieważ ten drugi dawał lepsze wyniki niż NSP w ich modelach. Ponadto SOP jest zadaniem trudniejszym niż NSP, ponieważ NSP można obejść, porównując tematykę dwóch fragmentów: jeśli się różnią, to jest mało prawdopodobne, aby drugi fragment następował po pierwszym.

**NSP kontra modelowanie języka**

BERT został zaprojektowany do kodowania dwukierunkowego, co oznacza, że BERT może przetwarzać zdania w obu kierunkach: zarówno od lewej do prawej, jak i od prawej do lewej (tj. analogicznie do dwukierunkowych LSTM). W związku z tym w zadaniach MLM i NSP można przekazać BERT całą sekwencję danych wejściowych naraz, a BERT uwzględni wszystkie słowa w tekście w swoich prognozach. Natomiast modelowanie języka w modelach opartych na GPT pozwala na przetwarzanie zdań tylko od lewej do prawej. Element maskującej uwagi w komponencie dekodera uniemożliwia modelowi przeglądanie kolejnego słowa w zdaniu i zmusza go do przewidywania kolejnego słowa.

**Przetwarzanie tokenów wejściowych w czasie rzeczywistym**

BERT ma możliwość dwukierunkowego przetwarzania, co może być korzystne podczas przetwarzania całych zdań wejściowych. Istnieją jednak sytuacje, w których prawdopodobnie konieczne będzie przetwarzanie sekwencji danych wejściowych w sposób sekwencyjny zamiast przetwarzania całej sekwencji. Na przykład rozpoznawanie mowy jest zazwyczaj wykonywane token po tokenie, aczkolwiek z co najmniej jednym zastrzeżeniem. W szczególności struktura języka niemieckiego umieszcza czasowniki posiłkowe na końcu zdania, a w języku japońskim tłumacze słuchają całych zdań przed tłumaczeniem na inny język, taki jak angielski. Innym zastosowaniem jest autouzupełnianie ciągu znaków lub przewidywanie następnego słowa. W pewnym sensie GPT-3 wykonuje „autouzupełnianie" na znacznie większą skalę: po otrzymaniu tekstowego monitu wejściowego, GPT-3 może generować całe zdania, akapity, a nawet artykuły.

**PODSUMOWANIE**

Ta sekcja rozpoczęła się od wprowadzenia do modelu BERT i jego niektórych funkcji. Zobaczyłeś, jak trenowany jest BERT i jakie są dostępne modele BERT. Następnie dowiedziałeś się, czym BERT różni się od wcześniejszych modeli przetwarzania języka naturalnego (NLP). Dodatkowo, nauczyłeś się, jak

przeprowadzać analizę NER (NER) za pomocą BERT, a także modelowania tematów (BERTopic) i tematów (TEMP). Następnie dowiedziałeś się, jak wykonywać zadania czyszczenia danych za pomocą BERT, takie jak normalizacja, oraz jak wyrażenia regularne są istotne w czyszczeniu danych w BERT. Następnie dowiedziałeś się o warstwach osadzania BERT, a także o osadzaniach tokenów, segmentów i pozycji. Zobaczyłeś przykład kodu w Pythonie, który wyjaśniał, jak utworzyć, trenować i zapisać model BERT. Następnie omówiliśmy niektóre wewnętrzne mechanizmy BERT, takie jak MLM, NSP i specjalne tokeny BERT. Poznałeś tokenizację podsłowną, BPE, WordPiece i SentencePiece. Ponadto dowiedziałeś się o podobieństwie zdań w BERT, kontekście wyrazów, a następnie o przykładach kodu Pythona, które generują tokeny BERT.