Ciąg dalszy następuje

*Filip GRALIŃSKI**

*Uniwersytet im. Adama Mickiewicza w Poznaniu

Temat dużych modeli języka (*Large Language Models*, *LLMs*), w szczególności najsłynniejszego ich przedstawiciela, czyli ChataGPT, jest ostatnio bardzo modny. Słyszymy o ich niezwykłych możliwościach, niektórych niepokoją pewne potencjalne zagrożenia z nimi związane.

Ciekawe, że modele języka opierają się na bardzo prostej zasadzie. W gruncie rzeczy są to po prostu uniwersalne „autouzupełniacze", które są w stanie dokończyć dowolny tekst, czasami z lepszym, czasami z gorszym skutkiem. W każdym razie mają „nawijkę", z którą żaden człowiek nie może się równać. Co więcej, zwykle nie planują one wiele kroków w przód, a jedynie przewidują następne słowo.

Prześledźmy działanie modelu języka na prostym przykładzie. Użyjemy otwartoźródłowego modelu Polish GPT-2. Jest to wprawdzie model gorszy niż komercyjny ChatGPT, ale możemy go łatwo uruchomić na swoim komputerze bez płacenia i pytania kogokolwiek o zgodę. Mamy też nad nim pełną kontrolę i możemy zajrzeć do jego „wnętrzności", co jest przydatne w poznawaniu zasady jego działania.

Powiedzmy, że chcielibyśmy poprosić model języka o dokończenie takiego prostego zdania:

Dzisiaj rano poszłem do piekarni i...

Czy właśnie napisałem *poszłem*?! Prawidłowa polszczyzna wymaga tutaj oczywiście formy *poszedłem*, ale krótsza forma jest, w gruncie rzeczy, częścią systemu (niestarannej) polszczyzny; dobry model języka powinien sobie radzić nie tylko z językiem pięknym i formalnym, lecz także z potocznym czy niechlujnym.

Zastanów się Czytelniku, jakie słowa mogą wystąpić dalej. Skonfrontujemy to później z przewidywaniami modelu.

Jak już wspomniałem, model w każdym kroku skupia się po prostu na przewidywaniu następnego wyrazu, a mówiąc ściślej, następnego **tokenu**. Token to taki wyrób wyrazopodobny, częste i krótkie wyrazy są zazwyczaj jednym tokenem, ale dłuższe i rzadsze mogą rozpaść się na dwa lub większą ich liczbę. Podział na tokeny, czyli **tokenizacja**, jest dość prostą mechaniczną procedurą. Na przykład początek naszego zdania po tokenizacji za pomocą modelu Polish GPT-2 będzie wyglądał tak:

Dzisiaj	rano	posz	łem	do	piekar	ni	i

Jak widać, nasze nieszczęsne *poszłem* rozpadło się na dwa tokeny, podobnie słowo *piekarni*. Zwróćmy jeszcze uwagę na to, że spacje doklejają się do następujących po nich wyrazach.

Co się dzieje dalej? Duże modele języka są przykładem szerszej klasy algorytmów nazywanych sieciami neuronowymi. Chwileczkę, czy jeśli mowa o neuronach, to ten artykuł nie powinien pojawić się w dziale biologicznym? Nie, nie, choć sieci neuronowe są luźno inspirowane tym, co wiemy o układzie nerwowym człowieka i innych organizmów, to nie są to żadne mózgi w słoikach. Sieci neuronowe są czysto matematycznym „ustrojstwem" – właściwie nie robią nic innego, tylko przemnażają i dodają dużo liczb.

No właśnie, skoro modele języka, będąc sieciami neuronowymi, mogą operować tylko na liczbach, oznacza to, że musimy w jakiś sposób przerzucić most między światem słów a światem liczb. Robimy to, **zanurzając** słowa (czy właściwie tokeny) w wielowymiarowej przestrzeni, a więc przypisując tokenom wektory, czyli ciągi liczb.

Co się dzieje dalej? Sieć neuronowa przykłada... kątomierz i mierzy kąty między tymi wektorami. Z tym kątomierzem to nawet nie jest bardzo duże uproszczenie i ubarwienie naszej opowieści – rzeczywiście mierzymy kąty między wektorami reprezentującymi słowa, w ten sposób sprawdzając, jak bardzo jedno słowo jest podobne do drugiego. Mierzenie kąta w zasadzie sprowadza się do liczenia iloczynu skalarnego między wektorami.

Karta graficzna to zasadniczo urządzenie służące – jak sama nazwa wskazuje – do rysowania dwu- i trójwymiarowych obrazów. Żeby narysować kosmitę, z którym walczymy w strzelance 3D, trzeba jednak wykonać mnóstwo przemnożeń wektorów i macierzy. Kilkanaście lat temu ktoś wpadł na pomysł, że kart graficznych można użyć właśnie w sztucznej inteligencji!

Tak więc do sieci neuronowej trafiają wektory reprezentujące kolejne tokeny, te wektory są przemnażane w kolejnych warstwach, również przez macierze, czyli „tabelki" liczb. Nie wchodząc w szczegóły, karta graficzna musi dokonać miliardów przemnożeń i dodawań.

No dobrze, zobaczmy w końcu, co model języka zrobi z naszym zdaniem. Skrypt w języku programowania Python, który uruchomi nasz model, jest tak krótki, że możemy go tutaj przytoczyć w całości.

Najpierw musimy załadować potrzebne biblioteki, tokenizator i właściwy model (linie 1–6). Jeśli nie mamy karty graficznej, model uruchomi się na zwykłym procesorze („cpu"), będzie trwało to wolniej, ale też zadziała.

```
1 from transformers import AutoTokenizer, AutoModelForCausalLM
2 import torch

3 model_name = 'sdadas/polish-gpt2-xl'
4 tokenizer = AutoTokenizer.from_pretrained(model_name)
5 device = 'cuda' if torch.cuda.is_available() else 'cpu'
6 model = AutoModelForCausalLM.from_pretrained(model_name).half().to(device)
```

Następnie tokenizujemy tekst, efekt widzieliśmy już wyżej (linie 7–8).

W końcu uruchamiamy nasz model (linie 9–10)...

...i wyświetlamy 10 najbardziej prawdopodobnych kontynuacji naszego tekstu (linie 11–14).

```
7  t = 'Dzisiaj rano poszłem do piekarni i'
8  tokens = tokenizer.encode(t)
9  out = model(torch.tensor(tokens).to(device))
10 probs = torch.softmax(out[0][-1], 0)
11 k = 10
12 top_values, top_indices = torch.topk(probs, k)
13 for p, ix in zip(top_values, top_indices):
14     print(tokenizer.decode(ix), p.item())
```

Najciekawsze, że model nie wskazuje po prostu kolejnego słowa, tylko **rozkład prawdopodobieństwa** na wszystkich możliwych tokenach-kontynuacjach. Oto lista 10 wypisanych tokenów wraz z prawdopodobieństwami:

kupiłem	0.321533203125	nie	0.0157623291015625
zobaczyłem	0.0263824462890625	poprosiłem	0.014801025390625
tam	0.02154541015625	w	0.011260986328125
chciałem	0.0174407958984375	widzę	0.01025390625
wziąłem	0.017303466796875	po	0.0101776123046875

Model zachował się całkiem rozsądnie, chyba Ty też, Czytelniku, przewidziałeś *kupiłem* jako najbardziej prawdopodobną kontynuację? Ale kolejne propozycje też są zupełnie sensowne.

Zwróćmy uwagę, że model musi mieć „zaszytą” gdzieś w sobie, w swoich wektorach i macierzach, całkiem sporą wiedzę, zarówno o języku (na przykład żeby nie zmienił w połowie zdania formy męskiej na żeńską albo pojedynczej na mnogą), jak i o świecie (co się robi w piekarni?).

Od modelu języka oczekujemy jednak, że wygeneruje nam dłuższy tekst, a nie tylko rozkład prawdopodobieństwa... Rozwiązanie jest proste: doklejamy token z najwyższym prawdopodobieństwem (*kupiłem*) do początkowego wejścia i po prostu uruchamiamy model jeszcze raz. Otrzymamy teraz taki rozkład:

sobie	0.19384765625	dwa	0.01904296875
chleb	0.08404541015625	świeże	0.0187530517578125
bu	0.031890869140625	3	0.0170745849609375
bułki	0.031402587890625	2	0.0161590576171875
dwie	0.020751953125	kilka	0.01425933837890625

...więc doklejamy *sobie* i jeszcze raz:

bu	0.1282958984375	dwie	0.018768310546875
chleb	0.0789794921875	kilka	0.01861572265625
bułki	0.06201171875	dro	0.0183258056640625
pą	0.033447265625	dwa	0.0176239013671875
świeże	0.0189208984375	bagie	0.016815185546875

Dlaczego *bu*?! A, zapewne to słowo *bułkę* ucięte w środku. Uruchommy model kolejny raz i sprawdźmy:

łkę	0.6474609375	łek	0.00867462158203125
łeczki	0.2005615234375	ł	0.004360198974609375
łe	0.10736083984375	ły	0.0034503936767578125
łka	0.011138916015625	łki	0.0014162063598632812
łę	0.00937652587890625	lki	0.0012111663818359375

Mieliśmy więc rację. A jak wygląda cały tekst dogenerowany przez model Polish GPT-2? Jeśli wykonamy 20 iteracji, to otrzymamy:

Dzisiaj	rano	posz	łem	do	piekar	ni	i	kupiłem	sobie	bu	łkę	z	szy	nką	i	serem	.	W	róciłem
do	domu	i	zjad	łem	.	Potem													

Taki mało ekscytujący tekst. Żeby jednak wiedzieć, jak go wygenerować, korzystając z modelu, potrzeba całkiem sporo matematyki: algebry liniowej, geometrii, teorii prawdopodobieństwa. Nie wspominając o uczeniu modelu, co wymaga „wciągnięcia” kolejnych działów matematyki i informatyki.

Rozwiązanie zadania M 1781.
Po podniesieniu do kwadratu obu stron równości $\sqrt{x}+\sqrt{y}=2\sqrt{2}$ i wykorzystaniu nierówności $x+y\geqslant 2\sqrt{xy}$ dostajemy

$$8=x+y+2sqrt(xy)\leqslant 2(x+y),$$

skąd $x+y\geqslant 4$. Ponadto z nierówności między średnią arytmetyczną a kwadratową mamy:

$$(*)\qquad x^2+y^2\geqslant\frac{1}{2}(x+y)^2,$$

więc $x^2+y^2\geqslant 8$. Ponownie korzystając z nierówności między średnią arytmetyczną a geometryczną, mamy:

$$64=\frac{2^{x^2+y}+2^{x+y^2}}{2}\geqslant\sqrt{2^{x^2+y}\cdot 2^{x+y^2}}=$$
$$=2^{\frac{1}{2}(x^2+y+x+y^2)}\geqslant 2^{\frac{1}{2}(4+8)}=2^6=64.$$

Zatem $x+y=4$ oraz $x^2+y^2=8$. Jednakże oznacza to, że w nierówności $(*)$ mamy równość, stąd $x=y=2$.

Zwróćmy uwagę – jakże różne rodzaje rozkładów prawdopodobieństwa otrzymujemy z modelu – czasami jest większa liczba kandydatów ze stosunkowo dużymi prawdopodobieństwami, a czasami mamy pewniaka, który bierze więcej niż połowę masy prawdopodobieństwa.