PyTorch w NLP

Od tokenizacji do predykcji kolejnego słowa

Przetwarzanie języka naturalnego, znane pod nazwą NLP (Natural Language Processing), od wielu lat pozostaje jednym z najciekawszych obszarów sztucznej inteligencji. To dziedzina, która próbuje opisać, zrozumieć i modelować język w formie, jaką posługują się ludzie, a następnie przekładać go na strukturę, którą można przetwarzać algorytmicznie. W praktyce NLP obejmuje szeroki zakres zagadnień, od analizy sentymentu i klasyfikacji tekstu po tłumaczenia maszynowe, streszczenia, automatyczne odpowiadanie na pytania i modele generatywne. Bez względu na złożoność tych zadań punktem wspólnym pozostaje konieczność zmiany języka naturalnego na liczby oraz odnalezienia regularności, które pozwalają maszynie przewidywać, jakie słowo pojawi się w tekście jako następne.

WPROWADZENIE

Choć współczesne systemy NLP kojarzą się przede wszystkim z ogromnymi modelami językowymi, w rzeczywistości ich działanie opiera się na kilku fundamentalnych koncepcjach. Tekst trzeba najpierw podzielić na mniejsze jednostki, zakodować w formie numerycznej, przetworzyć na sekwencje, które model będzie w stanie analizować, a następnie wykorzystać sieć neuronową, aby wyuczyć w jej wagach zależności pomiędzy kolejnymi elementami języka. Dopiero z tych podstaw powstają bardziej zaawansowane architektury, które potrafią pisać długie artykuły, prowadzić rozmowę lub rozwiązywać skomplikowane problemy.

W kontekście takich zadań **PyTorch** wypracował sobie bardzo mocną pozycję wśród narzędzi wykorzystywanych przez inżynierów i badaczy. Jego główną zaletą jest dynamiczny charakter pracy, który pozwala traktować budowę modelu jak proces programistyczny, a nie konfigurację zamkniętą w sztywnych ramach. Dzięki temu PyTorch nadaje się do eksperymentowania, modyfikowania architektury, testowania różnych wariantów danych i prowadzenia analiz, które są niezbędne w pracy z językiem. Kod napisany w PyTorch zachowuje się intuicyjnie, można go debugować krok po kroku, przerywać, przebudowywać fragmenty logiki modelu i sprawdzać w czasie rzeczywistym, jak zmienia się sposób jego działania. W NLP, gdzie każdy korpus danych ma inną charakterystykę, a jakość modelu zależy od wielu niuansów, taka elastyczność staje się ogromnym atutem.

W centrum tego artykułu znajduje się idea modeli językowych, czyli konstrukcji, które uczą się przewidywać kolejne słowo na podstawie dotychczasowego kontekstu. Na poziomie koncepcyjnym model językowy stara się odpowiedzieć na pytanie: jakie słowo jest najbardziej prawdopodobne po tym, które już „widział”? Choć z perspektywy człowieka może to wyglądać jak intuicyjne rozumienie treści, dla modelu jest to efekt uczenia się statystycznych regularności obecnych w tekście. Jeśli w korpusie często występuje połączenie „deep learning”, **model zacznie kojarzyć**, że po słowie „deep” z dużym prawdopodobieństwem pojawi się „learning”. W mniejszej skali, którą będziemy analizować, model będzie próbował uchwycić lokalną strukturę językową, opartą na zaobserwowanych fragmentach zdań.

Celem tego artykułu jest przeprowadzenie przez cały proces stworzenia prostego modelu językowego w PyTorch, od momentu, w którym posiadamy zwykły plik tekstowy, aż po sytuację, w której model potrafi samodzielnie wygenerować kolejne słowo. Taki model nie będzie rywalizował z dużymi sieciami transformatorowymi, ale pozwoli zrozumieć wszystkie mechanizmy, na których opierają się współczesne systemy NLP: tokenizację, budowę słownika, reprezentację słów w przestrzeni wektorowej, sekwencje treningowe, działanie warstw embedding, rekurencyjne przetwarzanie kontekstu i wreszcie sam proces generowania tekstu na podstawie przewidywanego rozkładu prawdopodobieństwa. Artykuł ma charakter praktyczny, ale jego celem nie jest jedynie przedstawienie konkretnego fragmentu kodu. Chodzi o zrozumienie logiki, która stoi za konstrukcją modelu językowego. Dopiero znajomość takich podstaw pozwala z większą świadomością podchodzić do bardziej skomplikowanych architektur, takich jak Transformery, które obecnie zaczynają dominować pod względem wydajności nad NLP. Warto doprecyzować, że mówimy o dominacji przede wszystkim w zadaniach generatywnych i pracy z długim kontekstem, gdzie mechanizm attention daje ogromną przewagę. W wielu wyspecjalizowanych zastosowaniach klasyczne podejścia NLP, takie jak modele statystyczne, sekwencyjne RNN, LSTM czy GRU, nadal są szybsze i bardziej efektywne obliczeniowo. Transformery nie zastąpiły całego tradycyjnego NLP, ale to one wyznaczają kierunek rozwoju tam, gdzie potrzebne jest modelowanie złożonych zależności i elastyczne rozumienie tekstu. Fundamentalna idea pozostaje jednak ta sama: przewidzieć następny token. Zrozumienie jej działania na małym przykładzie pozwala później lepiej interpretować zachowanie modeli, które liczą miliardy parametrów. W dalszej części artykułu omówimy przygotowanie środowiska, danych oraz podstawowych struktur, które będą potrzebne do implementacji prostego modelu językowego. To właśnie z tych elementów zbudujemy niewielką sieć neuronową zdolną do generowania tekstu w oparciu o to, czego nauczyła się z dostarczonego korpusu.

PRZYGOTOWANIE ŚRODOWISKA

Znaczenie przygotowania środowiska w projektach NLP

Praca nad modelem językowym nawet tak niewielkim, jak ten, który stworzymy w tym artykule, wymaga odpowiedniego przygotowania środowiska. Choć może to brzmieć jak techniczny szczegół, w praktyce od tego etapu w dużej mierze zależy późniejsza wygoda pracy, możliwość czystego eksperymentowania oraz stabilność całego procesu. NLP ma tę szczególną właściwość, że znaczenie drobnych zmian w danych czy konfiguracji modelu często okazuje się większe, niż się na początku wydaje. Wystarczy zmiana wersji tokenizatora, inny znak końca zdania, inna metoda podziału danych lub modyfikacja długości sekwencji wejściowych, aby cały eksperyment zachowywał się inaczej. Czasem w sposób trudny do wytłumaczenia, jeśli środowisko nie było odpowiednio izolowane i kontrolowane, dlatego środowisko powinno być przewidywalne, spójne oraz pozwalające na wielokrotne powtarzanie eksperymentów bez obaw o nieoczekiwane efekty uboczne. **Modele NLP są wrażliwe na każdy drobiazg**, nawet na biblioteki odpowiedzialne za kodowanie znaków czy sposób działania funkcji `split()`, która może zachowywać się inaczej w zależności od wersji Pythona lub systemu operacyjnego, w praktyce różnice mogą wynikać także z używanej powłoki np. bash, zsh, ponieważ niektóre narzędzia i skrypty potrafią działać inaczej w zależności od środowiska uruchomieniowego. W projektach komercyjnych problemy tego typu często ujawniają się dopiero wtedy, gdy model trafia na serwer produkcyjny, gdzie środowisko różni się od środowiska lokalnego.

W przeciwieństwie do narzędzi, które narzucają użytkownikowi konkretne procedury, PyTorch pozostawia dużą swobodę. Można uruchamiać go w prostym środowisku konsolowym, w Jupyter Notebookach, w kontenerze Dockera, w VS Code Remote Containers albo na pełnoprawnym serwerze obliczeniowym wyposażonym w GPU. Każdy z tych wariantów ma swoje zalety. Jupyter Notebook pozwala analizować tensory na bieżąco, Docker znacząco ułatwia zachowanie spójnego środowiska między różnymi maszynami, a GPU zwykle zapewnia większą wydajność w zadaniach obliczeniowych. Warto jednak pamiętać, że realna wydajność zależy nie tylko od samego układu, ale także od wersji sterowników, konfiguracji środowiska oraz różnic między generacjami kart, które potrafią mieć bardzo duży wpływ na osiągi. Z kolei zwykły terminal pozostaje najprostszym i najlżejszym rozwiązaniem, które nie wprowadza zbędnych warstw. W naszym przypadku nie ma potrzeby używania zaawansowanej infrastruktury. Wręcz przeciwnie, celowo pozostaniemy przy podstawowej konfiguracji, ponieważ pozwala ona lepiej zrozumieć każdy element procesu. Modele NLP mają to do siebie, że niezrozumienie jednego etapu przetwarzania danych skutkuje błędną interpretacją wyników, dlatego praca na prostej konfiguracji pomaga zachować pełną transparentność. Środowisko, jakie przygotowujemy teraz, będzie typowym laboratorium do eksplorowania modeli NLP. Będzie miejscem, w którym można swobodnie testować różne strategie tokenizacji, sposoby tworzenia słownika, długość sekwencji wejściowych czy nawet architekturę samej sieci, od najprostszych LSTM po bardziej złożone GRU i Transformers. Tak jak wcześniej wspomniałem, w NLP szczegóły bywają kluczowe, często drobna zmiana w sposobie przygotowywania danych potrafi całkowicie zmienić jakość uczenia modelu. Na przykład: jeśli zapomnimy o lowercasingu, model potraktuje słowa „Machine" i „machine" jako dwa różne byty, co może zaburzyć proces uczenia. Jeśli pominiemy znak kropki podczas tokenizacji, zniknie informacja o końcu zdania, co zmienia strukturę sekwencji. Dlatego tak ważne jest, aby środowisko uruchomieniowe, wraz z całym procesem przygotowania danych, było stabilne, przewidywalne i zarządzane w sposób kontrolowany.

Instalacja PyTorch i pierwsze testy

PyTorch, jako framework open-source rozwijany przez Meta AI oraz szeroką społeczność badaczy, stał się jednym z najważniejszych narzędzi w uczeniu maszynowym. Jego największą zaletą jest to, że nie próbuje ukrywać wewnętrznego działania modeli, jak robią to bardziej wysokopoziomowe biblioteki typu Keras. W PyTorch każdy tensor, gradient i operacja backpropagation jest w pełni widoczna, a dla osób chcących zrozumieć modele językowe od środka jest to absolutnie kluczowe. Instalacja PyTorch jest prosta i można ją przeprowadzić na kilka sposobów. Najczęściej używa się menedżera pip, ponieważ jest dostępny na każdej platformie i nie wymaga dodatkowych narzędzi. Warto jednak pamiętać, że PyTorch jest dystrybuowany w różnych wariantach: dla CPU, GPU z różnymi wersjami CUDA oraz z backendami takimi jak ROCm dla kart AMD. Jeśli komputer nie ma dedykowanej karty graficznej obsługującej CUDA, wystarczy podstawowa wersja działająca wyłącznie na CPU. Do treningu niewielkich modeli językowych CPU w zupełności wystarcza. Czasy treningu różnią się o sekundy, nie minuty czy godziny. Różnica pojawia się dopiero przy dużych korpusach, większych architekturach lub modelach Transformer.

Listing 1. Instalacja PyTorch

```
pip install torch torchvision torchaudio
```

Po instalacji warto przeprowadzić prosty test, który potwierdzi, że środowisko działa prawidłowo. Weryfikujemy wersję biblioteki, dostępność GPU oraz podstawowe działanie tensorów.

Listing 2. Test działania PyTorch

```
python3 - <<'EOF'
import torch
print("Wersja PyTorch:", torch.__version__)
print("GPU dostępne:", torch.cuda.is_available())
EOF
```

Listing 3. Typowy wynik

```
Wersja PyTorch: 2.3.0
GPU dostępne: False
```

Listing 4. Przykładowy wynik w przypadku wykrycia GPU

```
GPU dostępne: True
NVIDIA RTX 4080
```

Choć te polecenia wydają się drobiazgami, doświadczenie uczy, że duża część problemów z modelami NLP wynika nie z samej architektury, lecz z błędów środowiskowych: niekompatybilnych wersji

Zamów w prenumeracie >>

ŚWIT POLSKIEGO IT · ARCHITEKT W KRAINIE ITIES

INDEX: 285358

www.programistamag.pl

Magazyn programistów i liderów zespołów IT

programista

6/2025 (121)

Cena 32.90 zł (w tym VAT 8%)

Czym zachwycić

C++ 26

PyTorch w NLP. Od tokenizacji do predykcji kolejnego słow

Wizualizacja kąta obrotu obiektu z wykorzystanie

Koniec bańki generatywnego AI

Nie żyje król, niech żyje król! O alternatywach go AutoMappera

Nowy numer już w empikach