Wprowadzenie do dużych modeli językowych

Laboratorium 1_2 Generowanie tekstu z GPT-2

1. Wprowadzenie

GPT-2 (Generative Pre-trained Transformer 2) to model językowy opracowany przez OpenAI, oparty na architekturze Transformer z mechanizmem uwagi (attention). Model jest trenowany w trybie autoregresyjnym — uczy się przewidywać kolejne tokeny na podstawie poprzednich. Dzięki temu GPT-2 potrafi generować spójne, kontekstowe fragmenty tekstu.

Kluczowe cechy GPT-2:

- Architektura: Decoder-only Transformer
- Rozmiary: GPT-2 small (117M), medium (345M), large (762M), XL (1.5B)
- Trening: predykcja kolejnego tokenu (causal language modeling)
- Zastosowania: generowanie tekstu, uzupełnianie, streszczenie, tłumaczenie

W bibliotece HuggingFace Transformers dostęp do GPT-2 jest bardzo prosty dzięki klasom pipeline, AutoModelForCausalLM i AutoTokenizer. W trakcie laboratorium będziemy korzystać z modeli dostępnych publicznie na HuggingFace Hub.

2. Przykłady implementacji

Przykład 1: Instalacja bibliotek i pierwsze generowanie

```
# Instalacja zależności - jupter notebook
# !pip3 install transformers torch sentencepiece
# torch>=2.9.0 błąd wczytywania bibliotek w systemach windows
# !pip3 install torch==2.8.0 torchvision==0.23.0 torchaudio==2.8.0
from transformers import pipeline

# Wczytanie modelu GPT-2 (pobierane automatycznie przy pierwszym uruchomieniu)
generator = pipeline('text-generation', model='gpt2')

# Generowanie tekstu
result = generator(
    'Artificial intelligence is transforming',
    max_new_tokens=80,
    num_return_sequences=2,
    temperature=0.8,
    do_sample=True
)
```

```
for i, seq in enumerate(result):
    print(f'Sekwencja {i+1}:')
    print(seq['generated_text'])
    print('-' * 50)
```

Przykład 2: Sterowanie parametrami generowania

```
from transformers import AutoModelForCausalLM, AutoTokenizer
import torch

model_name = 'gpt2'
tokenizer = AutoTokenizer.from_pretrained(model_name)
model     = AutoModelForCausalLM.from_pretrained(model_name)

prompt = 'The future of renewable energy'
inputs = tokenizer(prompt, return_tensors='pt')

# Greedy decoding (deterministyczne, powtarzalne)
greedy = model.generate(**inputs, max_new_tokens=40)
print('Greedy:', tokenizer.decode(greedy[0], skip_special_tokens=True))

# Top-k sampling
topk = model.generate(**inputs, max_new_tokens=40,
                      do_sample=True, top_k=50, temperature=0.9)
print('Top-k:', tokenizer.decode(topk[0], skip_special_tokens=True))

# Top-p (nucleus) sampling
topp = model.generate(**inputs, max_new_tokens=40,
                      do_sample=True, top_p=0.92, temperature=0.8)
print('Top-p:', tokenizer.decode(topp[0], skip_special_tokens=True))

# Beam search
beam = model.generate(**inputs, max_new_tokens=40,
                      num_beams=5, early_stopping=True)
print('Beam:', tokenizer.decode(beam[0], skip_special_tokens=True))
```

Przykład 3: Generowanie z polskim modelem (Bielik / multilingual)

```
# Alternatywa: wielojęzyczny model GPT
# Opcja 1: GPT-2 multilingual fine-tuned (lekki)
# Opcja 2: facebook/opt-350m, bigscience/bloom-560m

from transformers import pipeline

# Bloom obsługuje wiele języków, w tym polski
gen = pipeline('text-generation', model='bigscience/bloom-560m')
```

```
result = gen(
    'Sztuczna inteligencja zmienia sposób',
    max_new_tokens=60,
    do_sample=True,
    temperature=0.85
)
print(result[0]['generated_text'])
```

Przykład 4: Analiza tokenizacji

```
from transformers import AutoTokenizer

tokenizer = AutoTokenizer.from_pretrained('gpt2')

texts = [
    'Hello, world!',
    'Artificial intelligence',
    'transformers library is great'
]

for text in texts:
    tokens = tokenizer.tokenize(text)
    ids    = tokenizer.encode(text)
    print(f'Tekst:   {text}')
    print(f'Tokeny:  {tokens}')
    print(f'IDs:     {ids}')
    print(f'Liczba tokenów: {len(tokens)}')
    print()
```

3. Zadania do samodzielnego rozwiązania

Każde zadanie powinno zostać wykonane z wykorzystaniem modelu o małej, średniej i dużej ilości parametrów.

Zadanie 1.1: Eksperyment z temperaturą

Napisz skrypt, który generuje 5 wersji tego samego promptu (np. 'Once upon a time in a small village') przy wartościach temperature: 0.1, 0.5, 0.8, 1.2, 1.5. Zapisz wyniki do pliku tekstowego i sformułuj wniosek: jak temperatura wpływa na kreatywność i spójność tekstu? Które wartości dają najlepsze rezultaty?

Zadanie 1.2: Porównanie strategii dekodowania

Zaimplementuj funkcję compare_decoding(prompt, model), która dla tego samego promptu generuje tekst czterema metodami: greedy, top-k (k=50), top-p (p=0.9) oraz beam search

(num_beams=4). Wyniki zebrać w słowniku. Dla każdej metody oblicz też liczbę unikalnych tokenów w odpowiedzi — która strategia daje najbardziej zróżnicowany tekst?

Zadanie 1.3: Generator chatbota

Zbuduj prostą aplikację konsolową, która działa w pętli: wczytuje prompt od użytkownika (input()), generuje odpowiedź GPT-2 i wyświetla ją. Dodaj do promptu krótki system prompt, np. 'You are a helpful assistant. User: {input}\nAssistant:'. Aplikacja powinna działać do wpisania 'quit'. Zadbaj o właściwe czyszczenie tokenów specjalnych.

Zadanie 1.4: Analiza statystyczna generowanych tekstów

Wygeneruj 10 różnych odpowiedzi na ten sam prompt. Dla każdej odpowiedzi oblicz: liczbę słów, liczbę zdań (po separatorze '.'), unikalność słownictwa (type-token ratio = unikalne_słowa / wszystkie_słowa). Wyniki przedstaw jako tabelę w konsoli i oblicz średnią oraz odchylenie standardowe każdej metryki.