Tokenizacja i jej wpływ na modele językowe
Lingwistyka Obliczeniowa | Laboratorium 2

Wojciech Bartoszek

1 Cel i zakres zadania

Celem laboratorium było zbadanie wpływu różnych strategii tokenizacji na efektywność i zachowanie prostego modelu językowego. Zaimplementowano i porównano trzy tokenizery: pretrenowany (GPT-2 BPE), własny tokenizer oparty na rozdzielaniu po białych znakach (whitespace) z ograniczonym słownikiem oraz tokenizer oparty na SentencePiece (BPE). Dla każdego tokenizera trenowano identyczną architekturę modelu (Transformer dekoder-only) i porównano je pod kątem metryk jakościowych (perplexity na poziomie słowa i znaku), statystyk OOV oraz metryk efektywności.

2 Tokenizery

Opis implementowanych tokenizerów oraz parametrów użytych w eksperymencie.

2.1 Pretrenowany tokenizer (GPT-2 BPE)

Użyto tokenizeru z biblioteki Hugging Face: `gpt2` (fast). Jeżeli tokenizer nie zawierał tokenu PAD, ustawiono go na token EOS, aby zapewnić spójność przy dopełnianiu.

2.2 Whitespace tokenizer (własny)

Implementacja dzieli tekst na tokeny według wyrażeń regularnych (`\w+` lub pojedyncze znaki interpunkcyjne). Słownik jest tworzony jako top-N najczęściej występujących tokenów w korpusie treningowym; tokeny rzadkie są mapowane na `<UNK>`. Specjalne tokeny: `<PAD>` i `<UNK>`.

2.3 SentencePiece (BPE)

SentencePiece trenowano na samodzielnie zbudowanym korpusie treningowym (1.2 MB) przy użyciu algorytmu BPE i tej samej wielkości słownika N jak w tokenizerze pretrenowanym.

3 Model i konfiguracja eksperymentu

Model: prosty Transformer dekoder-only zaimplementowany w module `modules/transformer.py`. Parametry użyte w eksperymencie (przykładowa, powtarzalna konfiguracja testowa):

- `vocab_size` = `VOCAB_SIZE` (ten sam dla wszystkich tokenizerów)
- `emb_dim` = 384
- `n_heads` = 6
- `n_layers` = 4

- `ff_dim` = 1536
- `max_seq_len` = 256
- `pad_token_id` = (zależne od tokenizeru)

Trening prowadzono strumieniowo na zbiorze `mikex86/stackoverflow-posts` z użyciem funkcji `train_streamed_lm`. Kluczowe ustawienia pętli treningowej (konfiguracja użyta w notatniku):

- `batch_size` = 16
- `max_length` = 256
- `steps_per_epoch` = 1000
- `num_epochs` = 5
- `lr` = 3e-4, warmup 200 kroków (skalowane względem akumulacji gradientu)
- `grad_clip` = 1.0, `grad_accum_steps` = 2

Urządzenie: automatyczny wybór `CUDA > MPS > CPU`. Ustawienia treningu i mierzenia czasu uwzględniają synchronizację dla CUDA i MPS.

4 Procedura oceny i metryki

Ocena każdej pary (tokenizer, model) obejmowała:

- obliczenie średniego token-level NLL (używając `evaluate_token_nll`),
- przeliczenie na perplexity na poziomie słowa i na poziomie znaku przez przeskalowanie NLL stosunkiem tokens/word i tokens/char (`word_and_char_perplexity`),
- statystyki OOV dla whitespace-tokenizera (liczba i procent OOV słów),
- metryki wydajności: przepustowość tokenów/s, ms/token dla generacji, ms/krok treningowego (moduł `modules/benchmark.py`),
- statystyki tokenizacji: średnia tokenów na słowo obliczona na próbie 1 MB nieużywanego w treningu tekstu oraz procent słów zakodowanych bez rozbicia lub zastąpienia `<UNK>`.

W raporcie znajdują się miejsca na wypełnienie wyników empirycznych (tabele i wykresy). Poniżej znajdują się przygotowane szablony.

5 Wyniki (miejsce na wyniki)

5.1 Podsumowanie metryk jakości

Poniżej przedstawiono wyniki jakościowe uzyskane po przeprowadzeniu eksperymentów.

Tokenizer	token-NLL	word-PPL	char-PPL	eval-tokens
Pretrained (GPT-2)	7.0913	1861.63	4.0257	407
Whitespace (top-N)	6.4189	613.31	3.2783	381
SentencePiece (BPE)	6.7868	1283.78	3.7583	404

Tabela 1: Porównanie jakości uzyskanych modeli

5.2 OOV dla whitespace tokenizer

Whitespace tokenizer używa słownika o rozmiarze N (takim samym jak wygenerowany z GPT-2). OOV liczono na tej samej próbce testowej co PPL. Wyniki:

- liczba OOV: 14
- łączna liczba słów: 421
- procent OOV: 3.3254%

5.3 Efektywność i tokenizacja

Tabela poniżej zawiera metryki wydajności oraz podstawowe statystyki tokenizacji obliczone na krótkiej próbce wykorzystanej do ewaluacji.

Tokenizer	vocab	tok/s	avg-tokens/word	pct-words-direct	tokens-measured
Pretrained (GPT-2)	50257	207659.51	1.0618	79.10%	894
Whitespace (top-N)	26428	121294.27	1.0000	96.67%	842
SentencePiece (BPE)	50257	135300.13	1.0546	95.01%	888

Tabela 2: Metryki wydajności i statystyki tokenizacji na krótkiej próbce

Metryki na buforze 1MB Dla stabilniejszej oceny tokenizacji użyto próbki 1MB (nieużywanej w treningu SP). Wyniki:

Tokenizer	tok/s (1MB)	tokens-measured	avg-tokens/word	direct-words	total-words
Pretrained (GPT-2)	191742.06	894	1.0618	333	421
Whitespace (top-N)	166216.59	842	1.0000	407	421
SentencePiece (BPE)	79057.18	888	1.0546	400	421

Tabela 3: Metryki tokenizacji i bezpośredniego kodowania na 1MB buforze testowym

5.4 Przykłady jakościowe

Poniżej trzy przykłady (każdy >=30 słów) wraz z tokenizacjami dla trzech tokenizerów.

Przykład 1

Tekst: In Python, list comprehensions provide a concise way to create lists. They are often faster than using loops, and they express intent clearly when mapping and filtering collections in everyday data processing tasks.

Pretrained Tokens:

```
['In', 'ĠPython', ',', 'Ġlist', 'Ġcomprehens', 'ions', 'Ġprovide', 'Ġa', 'Ġconcise',
 'Ġway', 'Ġto', 'Ġcreate', 'Ġlists', '.', 'ĠThey', 'Ġare', 'Ġoften', 'Ġfaster',
 'Ġthan', 'Ġusing', 'Ġloops', ',', 'Ġand', 'Ġthey', 'Ġexpress', 'Ġintent',
 'Ġclearly', 'Ġwhen', 'Ġmapping', 'Ġand', 'Ġfiltering', 'Ġcollections', 'Ġin',
 'Ġeveryday', 'Ġdata', 'Ġprocessing', 'Ġtasks', '.']
```

Whitespace Tokens:

```
['In', 'Python', ',', 'list', 'comprehensions', 'provide', 'a', 'concise', 'way',
'to', 'create', 'lists', '.', 'They', 'are', 'often', 'faster', 'than', 'using',
'loops', ',', 'and', 'they', 'express', 'intent', 'clearly', 'when', 'mapping',
'and', 'filtering', 'collections', 'in', 'everyday', 'data', 'processing', 'tasks',
 '.']
```

SentencePiece Tokens:

```
['_In', '_Python', ',', '_list', '_comprehensions', '_provide', '_a', '_concise',
 '_way', '_to', '_create', '_lists', '.', '_They', '_are', '_often', '_faster',
 '_than', '_using', '_loops', ',', '_and', '_they', '_express', '_intent',
 '_clearly', '_when', '_mapping', '_and', '_filtering', '_collections', '_in',
 '_everyday', '_data', '_processing', '_tasks', '.']
```

Przykład 2

Tekst: The quick brown fox jumps over the lazy dog, while the curious cat watches from the windowsill, pondering why humans keep typing this sentence to test keyboards and fonts across different systems.

Pretrained Tokens:

```
['The', 'Ġquick', 'Ġbrown', 'Ġfox', 'Ġjumps', 'Ġover', 'Ġthe', 'Ġlazy', 'Ġdog',
',', 'Ġwhile', 'Ġthe', 'Ġcurious', 'Ġcat', 'Ġwatches', 'Ġfrom', 'Ġthe',
'Ġwindows', 'ill', ',', 'Ġpond', 'ering', 'Ġwhy', 'Ġhumans', 'Ġkeep',
'Ġtyping', 'Ġthis', 'Ġsentence', 'Ġto', 'Ġtest', 'Ġkeyboards', 'Ġand',
'Ġfonts', 'Ġacross', 'Ġdifferent', 'Ġsystems', '.']
```

Whitespace Tokens:

```
['The', 'quick', 'brown', 'fox', 'jumps', 'over', 'the', 'lazy', 'dog',
',', 'while', 'the', 'curious', 'cat', 'watches', 'from', 'the', '<UNK>',
 ',', '<UNK>', 'why', 'humans', 'keep', 'typing', 'this', 'sentence',
  'to', 'test', 'keyboards', 'and', 'fonts', 'across', 'different',
   'systems', '.']
```

SentencePiece Tokens:

```
['_The', '_quick', '_brow', 'n', '_fox', '_jumps', '_over', '_the', '_lazy', '_dog',
 ',', '_while', '_the', '_curious', '_cat', '_watches', '_from', '_the', '_windows',
  'ill', ',', '_pon', 'dering', '_why', '_humans', '_keep', '_typing', '_this',
  '_sent', 'ence', '_to', '_test', '_keyboards', '_and', '_fonts', '_across',
  '_different', '_systems', '.']
```

Przykład 3

Tekst: When training language models, tokenization choices can greatly affect performance, memory usage, and generalization; understanding subword algorithms and OOV behavior helps practitioners make informed trade-offs for their applications.

Pretrained Tokens:

```
['When', 'Ġtraining', 'Ġlanguage', 'Ġmodels', ',', 'Ġtoken', 'ization', 'Ġchoices',
 'Ġcan', 'Ġgreatly', 'Ġaffect', 'Ġperformance', ',', 'Ġmemory', 'Ġusage', ',',
 'Ġand', 'Ġgeneral', 'ization', ';', 'Ġunderstanding', 'Ġsub', 'word',
 'Ġalgorithms', 'Ġand', 'ĠOO', 'OV', 'Ġbehavior', 'Ġhelps', 'Ġpractitioners',
 'Ġmake', 'Ġinformed', 'Ġtrade', '-', 'offs', 'Ġfor', 'Ġtheir',
  'Ġapplications', '.']
```

Whitespace Tokens:

```
['When', 'training', 'language', 'models', ',', '<UNK>', 'choices', 'can',
'greatly', 'affect', 'performance', ',', 'memory', 'usage', ',', 'and',
'<UNK>', ';', 'understanding', '<UNK>', 'algorithms', 'and', '<UNK>',
'behavior', 'helps', '<UNK>', 'make', 'informed', 'trade', '-', 'offs',
 'for', 'their', 'applications', '.']
```

SentencePiece Tokens:

```
['_When', '_training', '_language', '_models', ',', '_token', 'ization',
 '_choices', '_can', '_greatly', '_affect', '_performance', ',', '_memory',
  '_usage', ',', '_and', '_general', 'ization', ';', '_understanding', '_sub',
   'word', '_algorithms', '_and', '_OO', 'V', '_behavior', '_helps', '_pract',
   'ition', 'ers', '_make', '_informed', '_trade', '-', 'offs', '_for',
   '_their', '_applications', '.']
```

6 Dyskusja

Krótka interpretacja oczekiwanych efektów:

- Pretrenowany tokenizer (BPE) zwykle ma niską średnią tokens/word dla angielskich tekstów i niewielką liczbę OOV-ów przy porównaniu do prostego whitespace-tokenizera.
- Whitespace tokenizer łatwo wykrywa słowa jako osobne jednostki, ale ze względu na ograniczony słownik może mieć wysoki udział OOV; to wpływa na jakość modelu i może zwiększać tokens/word przez zamianę na `<UNK>` lub pozostawianie niepodzielonych wieloczłonowych tokenów.
- SentencePiece (BPE) zwykle łączy zalety obu podejść: umiarkowana liczba tokenów/word, niskie OOV dzięki subwordom, i przewidywalny słownik do eksperymentów.

7 Wnioski i dalsze prace

Wnioski po analizie porównawczej należy wypełnić na podstawie otrzymanych wyników. Potencjalne kierunki rozszerzeń:

- analizować wpływ różnych wielkości słownika N (np. 8k, 16k, 32k),
- testować warianty SentencePiece (unigram vs BPE),
- powtarzać eksperymenty na innych zbiorach (np. Wikipedia) oraz z większymi modelami, aby sprawdzić skalowalność obserwacji,