Implementacja transformera oraz eksperymenty z wariantami mechanizmu uwagi (Performer, Reformer) w zadaniach klasyfikacji tekstu

Dokumentacja projektu: część IV

Bartłomiej Borycki, Michał Iwaniuk

22 grudnia 2025

1 Testy jednostkowe

Testy są uruchamiane przy użyciu `pytest`. Dla izolacji i kontroli zależności stosowany jest mechanizm `monkeypatch`.

- **Mockowanie usług:** W testach loggera W&B wykorzystywany jest obiekt zastępczy (`DummyRun`) oraz `monkeypatch` do podmiany `wandb.init`. Dzięki temu testy nie wykonują połączeń sieciowych.
- **Izolacja I/O:** Zapisy checkpointów, CSV i sztucznych datasetów wykonywane są do katalogów tymczasowych (`tmp_path`).

1.1 Zakres testów

Testy weryfikują poprawność działania całego pakietu `src/textclf_transformer`. Testy są zorganizowane w katalogu `tests/unit/` zgodnie z konwencją nazewnictwa `pytest` (`test_*.py`). Raport pokrycia znajduje sie w tabeli **??**. Testy obejmują następujące obszary:

- **Logger** (`tests/unit/logger`):
  - Poprawność zapisu metryk do plików CSV (przy wyłączonym W&B) oraz logowanie do serwisu W&B.
  - Rejestrowanie metryk systemowych (np. zużycie pamięci GPU).
- **Tokenizer** (`tests/unit/tokenizer`):
  - Walidacja ładowania i treningu tokenizera z plików.
  - Poprawność kodowania tekstów (pojedynczych oraz z ramek Pandas) i obsługa etykiet.
  - Mechanizm maskowania wejścia dla modelu MLM (z pominięciem tokenów specjalnych).
- **Training** (`tests/unit/training` oraz `training/utils`):
  - **Pętla i skrypty:** Logika pętli treningowej dla klasyfikacji (zapis najlepszego modelu) i MLM, wznawianie treningu (*resume*) oraz tryb finetuningu (ładowanie wag pre-trained).
  - **Narzędzia:** Rozwiązywanie ścieżek względem root repozytorium, obsługa konfiguracji YAML, zapis i odczyt checkpointów (pełnych stanów oraz samych wag).
  - **Metryki i dane:** Obliczanie metryk (perplexity dla MLM, metryki sklearn dla klasyfikacji), poprawne działanie `collate_fn` (padding, przycinanie) oraz załadowanie `TensorDataset`.
- **Modele - ogólne i bloki** (`tests/unit/models`, `blocks`):
  - Propagacja maski paddingu w modelu Transformer i poprawność kształtów tensorów wyjściowych.
  - Struktura i inicjalizacja bloków MLP, Attention oraz EncoderBlock (mechanizmy rezydualne i normalizacja).
  - Współdzielenie wag (weight tying) w modelu MLM.
- **Mechanizmy atencji** (`tests/unit/models/attention`):
  - **MHA:** Zgodność numeryczna z implementacją PyTorch, obsługa `SDPA`, determinizm w trybie ewaluacji.

  - **FAVOR+:** Stabilność numeryczna wariantów funkcji ϕ (w tym `exp`), obsługa masek, poprawność gradientów i mechanizmów stabilizacji (global shift).
  - **LSH:** Respektowanie masek (padding i chunking), stabilność haszowania oraz weryfikacja analityczna na małych próbkach.

- **Embeddingi** (`tests/unit/models/embeddings`):
  - Poprawność wzorów kodowania pozycyjnego: sinusoidalne, uczone (1D) oraz RoPE (rotacja, cache, obsługa `position_ids`).
  - Inicjalizacja embeddingów tekstowych, zerowanie wektora paddingu.

- **Pooling i Heady** (`tests/unit/models/pooling`, `heads`):
  - **Pooling:** Poprawność algorytmów CLS, Mean, Max, Min.
  - **Heady:** Architektury poolerów w głowicach klasyfikacyjnych (BERT/RoBERTa) oraz struktura głowicy MLM.

Tabela 1: Raport pokrycia kodu testami

Moduł	Stmts	Miss	Cover
Core			
`__init__`	5	0	100%
Logger			
`__init__`	1	0	100%
`wandb_logger`	98	16	84%
Tokenizer			
`__init__`	1	0	100%
`wordpiece_tokenizer_wrapper`	117	8	93%
Training			
`__init__`	2	0	100%
`train`	52	9	83%
`training_loop`	238	41	83%
`utils/__init__`	4	0	100%
`utils/config`	56	28	50%
`utils/dataloader_utils`	42	1	98%
`utils/metrics_utils`	42	0	100%
`utils/train_utils`	59	1	98%
Modele – ogólne i bloki			
`__init__`	13	0	100%
`consts`	2	0	100%
`transformer`	40	10	75%
`transformer_classification`	29	3	90%
`transformer_mlm`	16	0	100%
`blocks/attention_block`	24	0	100%
`blocks/mlp_block`	10	0	100%
`blocks/transformer_encoder_block`	12	0	100%
Mechanizmy atencji			
`multihead_sdp_self_attention`	79	4	95%
`multihead_favor_self_attention`	180	13	93%
`multihead_lsh_self_attention`	145	3	98%
Embeddingi			
`positional_encodings`	24	0	100%
`rotary`	27	0	100%
`text_embeddings`	57	1	98%
Pooling i Heady			
`pooling`	37	1	97%
`classifier_head`	26	0	100%
`mlm_head`	22	1	95%
TOTAL	**1460**	**140**	**91%**

Liczba testów: 115 | Błędy: 0 | Niepowodzenia: 0 | Pominięto: 0 | Czas: 2.341s

2 Plan eksperymentów

Głównym celem przeprowadzonej części badawczej jest ewaluacja różnych wariantów mechanizmu atencji w architekturach typu *encoder-only* (BERT-like). Kluczowym aspektem analizy jest zbadanie zależności ("trade-off") pomiędzy jakością modelu w zadaniu klasyfikacji, kosztami treningu (zajętość pamięci VRAM i czas) oraz wydajnością inferencji. Dążymy do znalezienia optymalnej konfiguracji modelu pod kątem zastosowań praktycznych, szczególnie w kontekście przetwarzania długich sekwencji.

2.1 Zbiory danych

Eksperymenty przeprowadzono na zróżnicowanych zbiorach danych, dobranych tak, aby sprawdzić zachowanie mechanizmów atencji przy różnych długościach sekwencji wejściowych (L). Charakterystykę zbiorów przedsta-

wiono poniżej:

- **Wikipedia (Pretrening):** Korpus ogólny wykorzystywany do nauki reprezentacji języka. Aby zoptymalizować koszt obliczeniowy pretreningu, zastosowano strategię mieszaną: 75% treningu odbywa się na sekwencjach o długości 128 tokenów, a pozostałe 25% na sekwencjach o długości 512 tokenów.
- **IMDB (Klasyfikacja):** Zbiór recenzji filmowych służący do analizy sentymentu. Reprezentuje zadania o standardowej długości kontekstu.
  Max seq len: 512 tokenów.
- **Hyperpartisan (Klasyfikacja):** Zbiór artykułów newsowych klasyfikujący stronniczość polityczną. Reprezentuje zadania o długim kontekście.
  Max seq len: 4096 tokenów.
- **ArXiv (Klasyfikacja):** Zbiór tekstów naukowych, wymagający analizy bardzo długich zależności w tekście.
  Max seq len: 16 384 tokenów.

2.2 Architektura

Oznaczenia

- l^t – liczba warstw enkodera (liczba bloków Transformer).
- d^h – rozmiar wektora ukrytego (*hidden size*)
- d^f – rozmiar warstwy pośredniej w feed-forward network (FFN). W klasycznym BERT zwykle wynosi $4 \cdot d^h$.
- h – liczba głowic w mechanizmie wielogłowicowej uwagi.
- $d^{q|k|v}$ – wymiar przestrzeni zapytań (*query*), kluczy (*key*) oraz wartości (*value*) w każdej głowicy uwagi. (W klasycznym BERT przyjmuje się zazwyczaj $d^q = d^k = d^v = \frac{d^h}{h}$)

Architektura Bazowa

Jako standardową architekturę małego modelu BERT przyjmiemy $\text{BERT}_{\text{SMALL}}$ wprowadzoną w artykule *Well-Read Students Learn Better: On the Importance of Pre-training Compact Models*

$\text{BERT}_{\text{SMALL}}$:

l^t	d^h	d^f	h	$d^{q\|k\|v}$
4	512	2048	8	512

2.3 Metodyka uczenia

Proces uczenia dla każdej konfiguracji modelu podzielono na trzy etapy:

1. **Pretrening:** Nauka modelu od zera na zbiorze Wikipedia (cel: MLM - Masked Language Modeling).
2. **TAPT (Task-Adaptive Pretraining):** Dotrenowanie modelu na danych nieetykietowanych ze zbioru docelowego (kontynuacja MLM).
3. **Finetuning:** Ostateczne strojenie modelu pod zadanie klasyfikacji nadzorowanej.

Hiperparametry użyte w poszczególnych etapach przedstawiono w Tabeli **??**. Pozostałe hiperparametry były wspolne miedzy wszystkimi etapami treningu i eksperymentami Tabela **??**.

Tabela 2: Hiperparametry dla poszczególnych etapów uczenia.

Etap	Zbiór danych	Epoki	Learning rate	Min lr ratio
Pretrening (MLM)	Wikipedia	10	5×10^{-4}	0.5
TAPT (MLM)	IMDB	15	2×10^{-4}	0.5
	Arxiv	2		
	Hyperpartisan	10		
Finetuning (klasyfikacja)	IMDB	8	3×10^{-5}	0.2
	Arxiv	2	2×10^{-5}	
	Hyperpartisan	4	1×10^{-5}	

Tabela 3: Wspólne hiperparametry dla wszystkich etapów trening.

Kategoria	Parametr	Wartość
Architektura	MLP dropout	0.1
	Embedding dropout	0.1
	Pozycje	RoPE (base=10000, scale=1.0)
	Attention dropout	0.0
	Attention out dropout	0.1
	Projection bias	true
MLM Head (Pretrening i TAPT)	tie_mlm_weights	true
	mask_p	0.15
	mask_token_p	0.8
	random_token_p	0.1
Classification Head (Finetuning)	pooler_type	bert
Trening	batch_size	32768/max_seq_len
	warmup_ratio	0.1
	weight_decay	0.01
	max_grad_norm	1.0
	loss	cross_entropy

2.4 Implementacja mechanizmu atencji i optymalizacja obliczeniowa

W ramach realizacji celów projektowych, biblioteka została wyposażona w autorską implementację mechanizmu *Scaled Dot-Product Attention* (SDPA). Implementacja ta ma walor edukacyjny i demonstracyjny, pozwalając na pełną transparentność obliczeń. Posiada ona jednak charakter "naiwny"– wymaga obliczania pełnej macierzy atencji o wymiarach $N \times N$, co skutkuje kwadratową złożonością pamięciową $O(N^2)$ i brakiem niskopoziomowych optymalizacji dla jąder CUDA.

W kontekście planowanych badań na zbiorach o długich sekwencjach (Hyperpartisan: 4096 tokenów, ArXiv: 16 384 tokenów), wykorzystanie naiwnej implementacji okazało się niemożliwe ze względu na ograniczenia pamięciowe akceleratorów (błędy *Out of Memory*) oraz nieakceptowalny czas treningu.

Aby umożliwić przeprowadzenie eksperymentów w rozsądnym czasie i zapewnić rzetelny punkt odniesienia, doimplementowano obsługę natywnej funkcji biblioteki PyTorch (`F.scaled_dot_product_attention`). Rozwiązanie to automatycznie wykorzystuje zoptymalizowane algorytmy, takie jak **Flash Attention**, które:

- Redukują złożoność pamięciową do liniowej $O(N)$ względem długości sekwencji (brak konieczności zapisu pełnej macierzy atencji).
- Wykorzystują kafelkowanie (ang. *tiling*) do optymalizacji dostępu do pamięci podręcznej GPU.
- Oferują wysoki stopień zrównoleglenia, znacznie przyspieszając obliczenia.

Uwaga: W związku z powyższym, we wszystkich opisanych w dalszej części pracy eksperymentach (zarówno w fazie E1, jak i E2), wariant oznaczony jako **MHA (Standard)** wykorzystuje tę zoptymalizowaną, natywną implementację (Flash Attention). Pozwala to na traktowanie wyników MHA jako silnego, przemysłowego punktu odniesienia dla badanych atencji przybliżonych (LSH i FAVOR).

3 Eksperymenty

3.1 Eksperyment 1: Optymalizacja finetuningu (BERT Small)

Pierwsza faza eksperymentów koncentruje się na wyznaczeniu optymalnych hiperparametrów procesu finetuningu dla bazowej architektury `bert-small`, wykorzystującej standardową atencję MHA. Przetestowano wpływ trzech czynników: liczby zamrożonych warstw enkodera, wartości dropoutu w głowicy klasyfikacyjnej oraz metody agregacji wektorów (pooling). Szczegółową przestrzeń poszukiwań (Grid Search) przedstawiono w Tabeli **??**.

Ze względu na odmienne właściwości poszczególnych zbiorów danych (m.in. drastyczne różnice w długości sekwencji oraz specyfikę domeny), optymalizacja ta przeprowadzana jest niezależnie (per dataset).

Tabela 4: Przestrzeń poszukiwań hiperparametrów w Eksperymencie E1

Hiperparametr	**Testowane wartości**
Liczba zamrożonych warstw (N_{freeze})	$\{0, 1, 2\}$
Dropout klasyfikatora (P_{drop})	$\{0.1, 0.2\}$
Metoda poolingu	{CLS token, Mean pooling}

3.1.1 Wyniki: IMDB

e1_pre_tl.png

(a) Train Loss Finetuning (IMDB)

asd.png

(b) Eval Loss Finetuning (IMDB)

abc.png

(c) F1 Macro Finetuning (IMDB)

Rysunek 1: Wykresy w celach poglądowych, poźniej zostaną zmienione/poprawione

Tabela 5: Wyniki F1-macro na zbiorze testowym (zbiór IMDB).

Pooling	N_{freeze}	P_{drop}	**F1-macro**
CLS	0	0.1	93.22
	0	0.2	93.22
	1	0.1	92.95
	1	0.2	92.67
	2	0.1	92.97
	2	0.2	92.93
Mean	0	0.1	92.90
	0	0.2	92.93
	1	0.1	92.99
	1	0.2	92.87
	2	0.1	92.83
	2	0.2	92.85

3.1.2 Wyniki: Hyperpartisian

Eksperymenty są w trakcie, wyniki zostaną przedstawione w milestone 5

3.1.3 Wyniki: ArXiv

Eksperymenty są w trakcie, wyniki zostaną przedstawione w milestone 5

3.1.4 Wyniki: Podsumowanie

Wyniki optmalizacji przedstawione w tabeli **??**. Parametry te będą używane przy finetuningu modeli w Eksperymentach 2 (sekcja **??**) i 3 (sekcja **??**)

Tabela 6: Optymalne hiperparametry warstwy wybrane w fazie E1. (Wyniki dla pozostałych zbiorów w umieścimy w dalszej części badań)

Zbiór danych	N_{freeze}	P_{drop}	**Pooling**
IMDB	2	0.2	Mean
Hyperpartisan	–	–	–
ArXiv	–	–	–

3.2 Eksperyment 2: Ewaluacja atencji przybliżonych (LSH i FAVOR)

W drugiej fazie porównanujemy warianty atencji **LSH** (Locality Sensitive Hashing) oraz **FAVOR** (Fast Attention Via positive Orthogonal Random features) zgodnie z tabelą **??**.

Tabela 7: Przeszukiwane hiperparametry dla poszczególnych mechanizmów atencji.

Atencja	**Hiperparametr**	**Testowane wartości**
LSH	Liczba haszy (N_{hashes})	$\{2, 4\}$
	Wielkość bloku ($Chunk$)	$\{64, 128\}$
FAVOR	Liczba losowych cech ($N_{features}$)	$\{0.125, 0.25, 0.5, 1.0\} \times d^{q\|k\|v}$

3.2.1 Wyniki: IMDB

Tabela 8: Wyniki F1-macro dla różnych konfiguracji mechanizmów atencji (zbiór IMDB).

Atencja	N_{hashes}	$Chunk$	**F1-macro**
LSH	2	64	-
	2	128	-
	4	64	-
	4	128	-
Atencja	$N_{features}$		**F1-macro**
FAVOR	0.125		91.16
	0.25		91.20
	0.5		91.70
	1.0		91.74

3.3 Eksperyment 3: Architektura AutoTinyBERTS1

W trzeciej fazie przeprowadzimy eksperymenty na architekturze AutoTinyBERTS1 opisanej w artykule *AutoTinyBERT: Automatic Hyper-parameter Optimization for Efficient Pre-trained Language Models*, aby sprawdzić, czy wnioski z tego artykułu przekładają się na inne mechanizmy atencji oraz na inne dane.