Fine-tuning vs. trening from scratch dla problemu klasyfikacji tekstu

Jacek Tyszkiewicz

December 14, 2025

1 Wprowadzenie i cel

Celem eksperymentu jest porównanie dwóch podejść do trenowania modeli językowych dla zadania **klasyfikacji pełnych tekstów**:

1. **Trening od zera (from-scratch)**: mały model decoder-only z losową inicjalizacją, trenowany wyłącznie na danych docelowych.
2. **Fine-tuning**: dostrajanie wstępnie wytrenowanego modelu w moim przypadku HerBERT na tym samym zbiorze danych.

W analizie porównano: jakość (accuracy, F1), przebieg uczenia (stabilność, zbieżność), przeuczenie (krzywe walidacyjne), czasy treningu i inferencji oraz rozmiar modeli (liczba parametrów).

2 Zbiór danych

2.1 Opis zbioru

Nazwa zbioru: `jziebura/polish_youth_slang_classification`
Źródło: Hugging Face Datasets (link)
Język: polski
Zadanie: klasyfikacja pełnych tekstów (ang. *text classification*) do klas sentymentu.

2.2 Statystyki i podziały

Użyto dostarczonych splitów: `train/validation/test` (bez tworzenia własnych podziałów).

Table 1: Podstawowe statystyki zbioru danych (po czyszczeniu)

Split	Liczba przykładów	Średnia długość (znaki)	Liczba klas
Train	4335	78.98	3
Validation	542	79.97	3
Test	543	81.21	3

Długość tekstów (w znakach) jest zróżnicowana - dla treningu, walidacji i dla testu.

2.3 Rozkład klas

Zbiór danych jest niezbalansowany: dominuje klasa 1, a najmniej przykładów ma klasa 2.

Split	Klasa 0	Klasa 1	Klasa 2	Suma	Dominuje	Najrzadsza
Train	1259 (29.1%)	2217 (51.1%)	859 (19.8%)	4335	1	2
Validation	155 (28.6%)	273 (50.4%)	114 (21.0%)	542	1	2
Test	148 (27.3%)	275 (50.6%)	120 (22.1%)	543	1	2

2.4 Przykłady i preprocessing

Na podstawie weryfikacji i prostych kroków czyszczenia stwierdzono, że zbiór jest czysty: nie wykryto istotnej liczby pustych rekordów ani problemów z kodowaniem (liczności splitów po filtracji praktycznie się nie zmieniły). Zastosowano jedynie **normalizację białych znaków** (zamiana wielu spacji na jedną i `strip`) oraz **standaryzację nazw kolumn**. Próba usunięcia duplikatów na podstawie pary (text, label) wykazała, że zbiór ich nie zawiera.

3 Tokenizacja i przygotowanie danych

3.1 Parametry tokenizacji

- Maksymalna długość sekwencji: **256** (ustawienie `max_len=256`).
- Strategia ucinania: `truncation=True`.
- Padding: **dynamiczny** do maksymalnej długości w batchu (padding realizowany w `collate_fn`).

3.2 Tokenizer: fine-tuning

- Model/tokenizer: `allegro/herbert-base-cased`.
- Typ tokenizera: **WordPiece**.
- Rozmiar słownika: `vocab_size` (zgodnie z konfiguracją tokenizera; wartość raportowana w `run_info.json`).

3.3 Tokenizer: from-scratch

- Tokenizer: `allegro/herbert-base-cased` (ten sam co dla fine-tuningu).
- Typ tokenizera: **WordPiece**.
- Rozmiar słownika: `vocab_size` (jak wyżej).
- Pochodzenie / trening: wykorzystano gotowy tokenizer z HerBERTa (tokenizer nie był trenowany od zera); **od zera trenowane były wagi modelu transformera oraz głowa klasyfikacyjna**.

4 Opis modeli

4.1 Model trenowany od zera (from-scratch)

4.1.1 Architektura

- Typ: **decoder-only**.
- Liczba warstw: **4** (`n_layers=4`).
- Liczba głów uwagi: **4** (`n_heads=4`).
- Wymiar embeddingu: **192** (`d_model=192`).
- Dropout: **0.2** (`dropout=0.2`).
- Parametry ogółem: **11.4M** (dokładnie: `11,417,475`).

4.1.2 Głowa klasyfikacyjna

- Reprezentacja: **mean pooling** po stanach ukrytych (z maskowaniem `attention_mask`).
- MLP: `Linear(d_model→d_model) + GELU + Dropout + Linear(d_model→num_labels)`.

4.2 Model po fine-tuningu

4.2.1 Wybór modelu

Wybrany model: `allegro/herbert-base-cased` (encoder-only, język polski).

Uzasadnienie wyboru (język, zadanie, dostępność, jakość baseline):

- Model jest wytrenowany na języku polskim, więc zapewnia sensowny baseline dla klasyfikacji tekstu PL.
- Jest łatwo dostępny w HuggingFace i dobrze wspierany w ekosystemie `transformers`.
- Daje znacznie lepsze wyniki niż model uczony od zera na małym zbiorze (kilka tys. przykładów).

4.2.2 Głowa klasyfikacyjna

- Użyta architektura: `AutoModelForSequenceClassification` (`BertForSequenceClassification` dla HerBERTa).
- Parametry głowy: wejście to wektor reprezentacji `[CLS]` (po wyjściu z encodera), następnie `Dropout` oraz warstwa liniowa `Linear(hidden_size -> num_labels)`. Dla HerBERTa `hidden_size = 768`, `num_labels = 3`.

4.2.3 Strategia fine-tuningu

- Pełny fine-tuning: **nie** (nie aktualizowano wszystkich wag modelu bazowego).
- Zamrożone warstwy: **tak** — zamrożony cały encoder HerBERT, trenowane tylko parametry adapterów LoRA oraz głowa klasyfikacyjna.
- LoRA: **tak**, $r = 8$, $\alpha = 16$, dropout $= 0.05$, target modules: `query`, `value` (w mechanizmie self-attention).

5 Ustawienia treningu

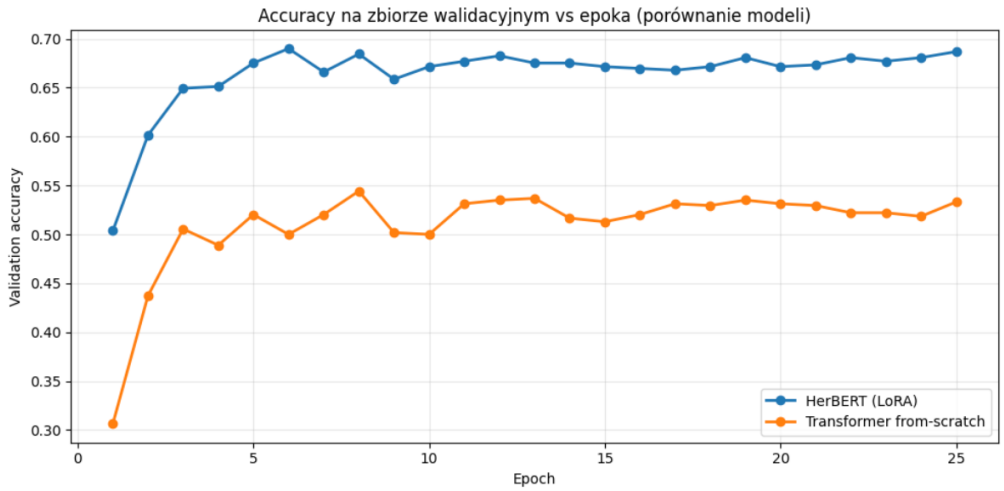
5.1 Wspólne elementy

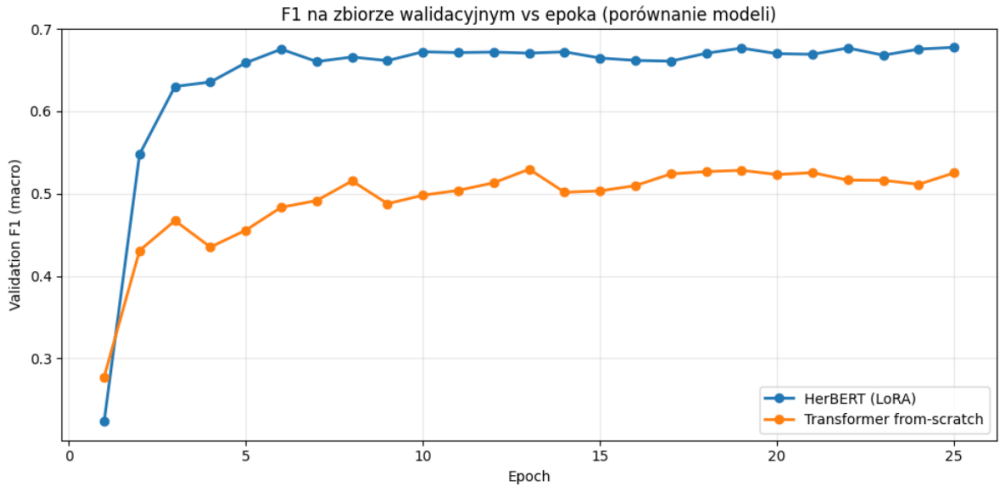
- Optymalizator: AdamW
- Funkcja straty: CrossEntropyLoss
- Gradient clipping: 1.0
- Scheduler/warmup: scheduler liniowy z warmup (`warmup_ratio`=0.06)
- Early stopping: brak (nie użyto)

5.2 Hiperparametry: fine-tuning

- learning rate: $2 \cdot 10^{-4}$
- liczba epok: 8
- weight decay: 0.01
- max_len (tokenizacja): 256
- precision: bf16

6 Metryki





6.1 Wyniki na zbiorze testowym

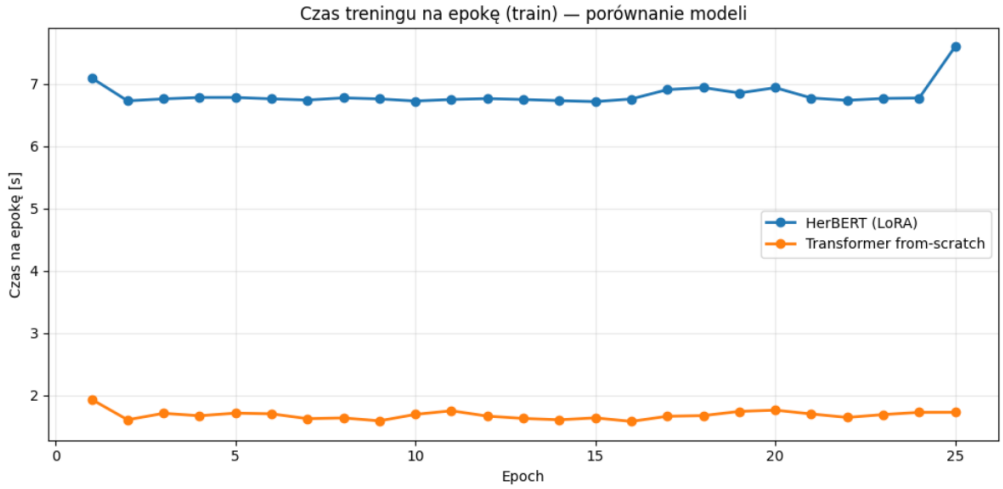
Model	Accuracy	F1 (macro)
HerBERT + LoRA	0.6906	0.6806
From-scratch (best.pt)	0.5249	0.5147

Table 2: Wyniki na zbiorze testowym.

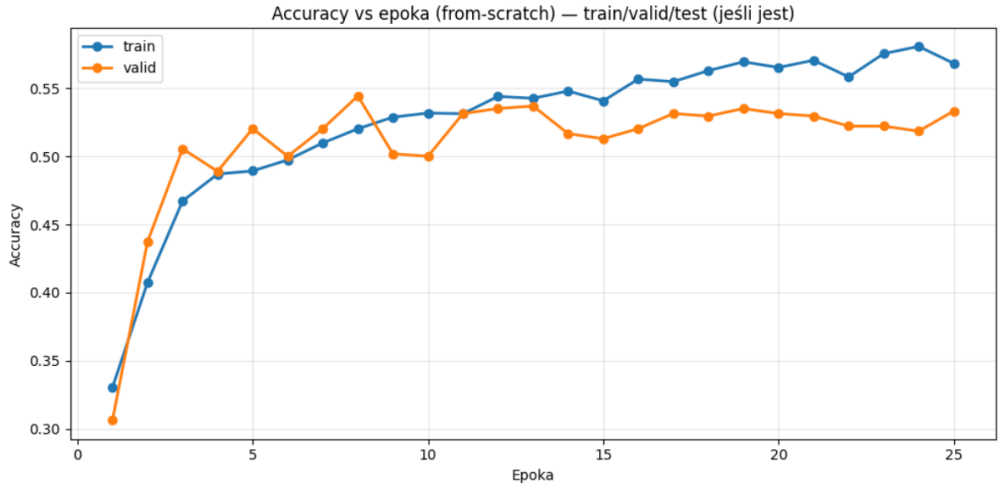
Na zbiorze walidacyjnym model **HerBERT (LoRA)** osiąga wyraźnie lepsze wyniki niż **Transformer trenowany od zera** zarówno dla accuracy, jak i F1 (macro). HerBERT szybko rośnie w pierwszych epokach i stabilizuje się na poziomie ok. **0.67–0.69** (accuracy) oraz **0.66–0.68** (F1). Model from-scratch uczy się wolniej i po kilku epokach wchodzi w plateau, utrzymując ok. **0.50–0.54** (accuracy) oraz **0.48–0.53** (F1). Wyniki wskazują, że pretrenowanie (HerBERT) daje szybszą zbieżność i lepszą generalizację.

Na zbiorze testowym obserwujemy analogiczny trend jak na walidacji: model **HerBERT + LoRA** osiąga wyraźnie lepsze wyniki niż model trenowany od zera. Uzyskano **accuracy = 0.6906** oraz **F1 (macro)**

= **0.6806** dla HerBERT + LoRA, natomiast dla modelu **from-scratch** odpowiednio **accuracy = 0.5249** i **F1 (macro) = 0.5147**. Różnica potwierdza, że fine-tuning modelu wstępnie wytrenowanego zapewnia lepszą generalizację na danych testowych.



Czas trenowania istotnie różni się między modelami. Dla **HerBERT (LoRA)** łączny czas treningu wyniósł **170.61 s**, co daje średnio **6.82 s/epokę**. Dla **Transformera trenowanego od zera** łączny czas treningu wyniósł **42.02 s**, czyli średnio **1.68 s/epokę**.



Model trenowany od zera osiąga szybki wzrost accuracy w pierwszych kilku epokach, co wskazuje na szybką zbieżność na początku treningu. Następnie metryka walidacyjna oscyluje w niewielkim zakresie, co sugeruje stabilne uczenie. Od okolic **12-13 epoki** obserwujemy stopniowy wzrost accuracy na zbiorze treningowym przy braku analogicznej poprawy na walidacji, a nawet niewielkim spadku, co wskazuje na lekkie **przeuczenie** (rosnąca luka między train i valid).

6.2 Rozmiar modelu

Na podstawie plików:

- **from-scratch:** 11 417 475 parametrów
- **fine-tuned (HerBERT + LoRA):** 124 742 406 parametrów (łącznie)
  w tym trenowalne (LoRA): 297 219 parametrów

6.3 Czas wnioskowania na zbiorze testowym

Model	inference_s_test [s]
Transformer from-scratch	0.11938
HerBERT (LoRA)	0.37919

Table 3: Czas wnioskowania (inference) na zbiorze testowym.

6.4 Analiza porównawcza

Fine-tuning zwykle przewyższa trening od zera, ponieważ model startuje z wagami wstępnie wyuczonymi na dużych zbiorach tekstu, co daje lepsze reprezentacje językowe już na początku uczenia. Dzięki temu potrzeba mniej danych i mniej epok, a zbieżność jest szybsza oraz stabilniejsza, co w praktyce często przekłada się na lepszą generalizację na nowych przykładach.

Trening od zera może być natomiast preferowany wtedy, gdy nasza domena bardzo mocno różni się od danych, na których trenowano model bazowy, czyli występuje duży domain shift. Ma to też sens, gdy dysponujemy bardzo dużą liczbą przykładów w swojej domenie oraz odpowiednimi zasobami obliczeniowymi, albo gdy zależy nam na pełnej kontroli nad architekturą i tokenizacją. Czasem powodem są również ograniczenia licencyjne lub prawne, które utrudniają wykorzystanie gotowego modelu wstępnie wytrenowanego.

Wrażliwość na rozmiar modelu i zbioru danych wygląda inaczej w obu podejściach. W fine-tuningu nawet przy mniejszym zbiorze można osiągać dobre wyniki, choć wraz ze wzrostem liczby parametrów rośnie ryzyko przeuczenia i potrzeba mocniejszej regularizacji. W treningu od zera sytuacja jest bardziej wymagająca: większy model zazwyczaj wymaga proporcjonalnie większej liczby danych, a przy zbyt małym zbiorze łatwo o słabe wyniki lub niestabilny trening.

Z perspektywy implementacji kluczowe jest monitorowanie krzywych uczenia (train/valid) i wybór najlepszego checkpointu zamiast trenowania „na sztywno" do końca (np. przez early stopping). W praktyce duży wpływ na stabilność ma dobór learning rate i rozgrzewki (warmup), a clipping gradientów pomaga ograniczać problemy z nagłymi skokami. Dodatkowo LoRA jest często dobrym kompromisem, bo pozwala uzyskać wysoką jakość przy mniejszym koszcie treningu. Dobrymi metrykami do porównania modeli są accuracy i F1, a także czas treningu, czas inferencji oraz liczbę parametrów.

Linki

Kod treningowy i tokenizacji (jeden skrypt): GitHub repozytorium

Skrypty ewaluacyjne: Google Drive

Literatura

1. Jacob Devlin, Ming-Wei Chang, Kenton Lee, Kristina Toutanova, *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*, arXiv:1810.04805, 2018. Dostęp: `https://arxiv.org/pdf/1810.04805`

2. Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, *Language Models are Unsupervised Multitask Learners*, 2019. Dostęp: `https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf`