Laboratorium 2 # Fine-tuning BERT do klasyfikacji tekstu

1. Wprowadzenie

BERT (Bidirectional Encoder Representations from Transformers) to model opracowany przez Google w 2018 roku. W odróżnieniu od modeli autoregresyjnych (GPT), BERT jest modelem encodera — analizuje cały tekst jednocześnie, uwzględniając kontekst zarówno z lewej, jak i z prawej strony każdego tokenu.

BERT jest wstępnie trenowany (pre-trained) na dwóch zadaniach:

- MLM (Masked Language Modeling) — przewidywanie zamaskowanych tokenów
- NSP (Next Sentence Prediction) — predykcja kolejności zdań

Fine-tuning polega na dodaniu do modelu BERT warstwy klasyfikacyjnej (linear head) i dotrenowaniu całego modelu na niewielkim, oznaczonym zbiorze danych. Dzięki wiedzy zawartej w wagach pre-treningowych, fine-tuning wymaga relatywnie małych danych i daje wysoką jakość.

W laboratorium użyjemy modeli BERT lub DistilBERT (wersja 40% lżejsza, 60% szybsza przy zachowaniu 97% jakości).

2. Przykłady implementacji

Przykład 1: Klasyfikacja sentymentu — szybki start z pipeline

```
# !pip3 install datasets scikit-learn
# wersja scikit-learn==1.8.0 nie działa poprawnie w windows
```

```
# !pip3 install scikit-learn==1.7.0
# w przypadku problemów z numpy, należy zainstalować wersję <2
```

```
from transformers import pipeline
```

```

```

```
# Gotowy pipeline do analizy sentymentu
```

```
classifier = pipeline('sentiment-analysis',
```

```
                      model='distilbert-base-uncased-finetuned-sst-2-english')
```

```

```

```
texts = [
```

```
    'I absolutely love this product, it works perfectly!',
```

```
    'This is the worst experience I have ever had.',
```

```
    'The movie was okay, nothing special.'
```

```
]
```

```

```

```
results = classifier(texts)
```

```
for text, res in zip(texts, results):
```

```
    print(f'{res["label"]:10s} ({res["score"]:.3f})  |  {text[:50]}')
```

Przykład 2: Fine-tuning DistilBERT na własnych danych

```
from transformers import (AutoTokenizer, AutoModelForSequenceClassification,
TrainingArguments, Trainer)
from datasets import Dataset
import numpy as np
from sklearn.metrics import accuracy_score, f1_score

# --- Przygotowanie danych (przykładowe) ---
train_data = {
    'text':  ['great product', 'terrible quality', 'it works fine',
              'horrible experience', 'highly recommended', 'waste of money'],
    'label': [1, 0, 1, 0, 1, 0]
}
train_dataset = Dataset.from_dict(train_data)

# --- Tokenizacja ---
model_name = 'distilbert-base-uncased'
tokenizer  = AutoTokenizer.from_pretrained(model_name)

def tokenize(batch):
    return tokenizer(batch['text'], truncation=True,
                     padding='max_length', max_length=128)

train_dataset = train_dataset.map(tokenize, batched=True)
train_dataset.set_format('torch', columns=['input_ids','attention_mask','label'])

# --- Model i trening ---
model = AutoModelForSequenceClassification.from_pretrained(model_name,
num_labels=2)

def compute_metrics(eval_pred):
    logits, labels = eval_pred
    preds = np.argmax(logits, axis=-1)
    return {'accuracy': accuracy_score(labels, preds),
            'f1':       f1_score(labels, preds)}

args = TrainingArguments(
    output_dir='./bert_output',
    num_train_epochs=3,
    per_device_train_batch_size=4,
    logging_steps=5,
    save_steps=50,
    no_cuda=True  # usuń jeśli masz GPU
)

trainer = Trainer(
```

```
    model=model,
    args=args,
    train_dataset=train_dataset,
    compute_metrics=compute_metrics
)

trainer.train()
```

Przykład 3: Klasyfikacja wieloklasowa z HuggingFace Datasets

```
from datasets import load_dataset
from transformers import AutoTokenizer, AutoModelForSequenceClassification
import torch

# Wczytanie publicznego zbioru (AG News: 4 klasy tematyczne)
dataset = load_dataset('ag_news', split='test[:500]')
classes = ['Świat', 'Sport', 'Biznes', 'Technologia']

tokenizer = AutoTokenizer.from_pretrained('textattack/bert-base-uncased-ag-news')
model     = AutoModelForSequenceClassification.from_pretrained(
                'textattack/bert-base-uncased-ag-news')
model.eval()

def predict(text):
    inputs  = tokenizer(text, return_tensors='pt',
                        truncation=True, max_length=256)
    with torch.no_grad():
        logits = model(**inputs).logits
    return classes[logits.argmax().item()]

# Przykładowe predykcje
for item in dataset.select(range(3)):
    pred = predict(item['text'])
    true = classes[item['label']]
    print(f'Prawda: {true:12s}  Predykcja: {pred}')
    print(f'  {item["text"][:80]}...')
    print()
```

3. Zadania do samodzielnego rozwiązania

Zadanie 2.1: Fine-tuning na zbiorze SST-2

Pobierz 200 przykładów ze zbioru sst2 (HuggingFace Datasets). Podziel na 160 treningowych i 40 testowych. Przeprowadź fine-tuning DistilBERT przez 3 epoki. Oblicz accuracy i F1-score na zbiorze testowym. Porównaj wyniki z klasyfikatorem regresji logistycznej na TF-IDF. Który model daje lepsze wyniki przy tak małym zbiorze?

Zadanie 2.2: Wizualizacja uwagi (attention)

Użyj modelu BERT z opcją output_attentions=True. Wczytaj zdanie 'The bank can guarantee deposits will eventually cover future tuition costs.' i zwizualizuj wagi uwagi dla warstwy 6, głowy 0. Użyj matplotlib (matshow) do pokazania macierzy uwagi. Co model 'widzi' jako ważne tokeny dla słowa 'bank'?

Zadanie 2.3: Klasyfikacja toksyczności komentarzy

Użyj modelu 'unitary/toxic-bert' do klasyfikacji listy 10 komentarzy (zdefiniuj własne, zarówno neutralne jak i potencjalnie toksyczne). Wyodrębnij prawdopodobieństwa poszczególnych kategorii (toxic, severe_toxic, obscene, insult, threat). Przedstaw wyniki jako tabelę. Zidentyfikuj próg odcięcia (threshold), przy którym precision i recall są zbalansowane.

Zadanie 2.4: Analiza wpływu rozmiaru danych treningowych

Przetestuj fine-tuning DistilBERT na 4 różnych rozmiarach zbioru treningowego: 50, 100, 200, 400 przykładów (z tego samego zbioru). Dla każdego rozmiaru zapisz F1-score na stałym zbiorze testowym. Narysuj wykres learning curve (matplotlib) zależności F1 od liczby przykładów treningowych.