Laboratorium 3 # LLM w zadaniu Question Answering (QA)

## 1. Wprowadzenie

Question Answering (QA) to jedno z kluczowych zadań NLP, polegające na automatycznym odpowiadaniu na pytania w języku naturalnym. Wyróżniamy dwa główne warianty:

- Extractive QA — model lokalizuje odpowiedź jako fragment (span) w podanym kontekście. Przykład: SQuAD, modele BERT fine-tuned.
- Generative QA (Open-Domain) — model generuje odpowiedź swobodnie, bez wskazanego kontekstu. Przykład: modele GPT, T5, FLAN-T5.
- Retrieval-Augmented Generation (RAG) — najpierw pobierane są relewantne dokumenty, następnie model generuje odpowiedź na ich podstawie.

W laboratorium skupimy się na extractive QA z BERT oraz generatywnym QA z FLAN-T5 (darmowy, otwarty model Google). FLAN-T5 jest wytrenowany na setkach zadań NLP i bardzo dobrze radzi sobie z odpowiadaniem na pytania bez dodatkowego fine-tuningu.

## 2. Przykłady implementacji

### Przykład 1: Extractive QA z BERT

```
from transformers import pipeline

qa = pipeline('question-answering',
              model='deepset/bert-base-cased-squad2')

context = '''
    Python is a high-level, general-purpose programming language.
    Its design philosophy emphasizes code readability, notably through
    significant indentation. Python was created by Guido van Rossum
    and first released in 1991. It supports multiple programming
    paradigms, including structured, object-oriented and functional.
'''

questions = [
    'Who created Python?',
    'When was Python first released?',
    'What does Python design philosophy emphasize?'
]

for q in questions:
    answer = qa(question=q, context=context)
    print(f'Q: {q}')
    print(f'A: {answer["answer"]}  (score: {answer["score"]:.3f})')
```

```
    print()
```

## Przykład 2: Generative QA z FLAN-T5

```
from transformers import pipeline

# FLAN-T5 — darmowy, otwarty model instruction-tuned Google
qa_gen = pipeline('text2text-generation',
                  model='google/flan-t5-base')

def ask(question, context=None):
    if context:
        prompt = f'Answer the question based on the context.\nContext:
{context}\nQuestion: {question}\nAnswer:'
    else:
        prompt = f'Answer the following question: {question}'
    result = qa_gen(prompt, max_new_tokens=100)
    return result[0]['generated_text']

# Bez kontekstu (wiedza z pre-treningu)
print(ask('What is the capital of France?'))

# Z kontekstem
ctx = 'The Eiffel Tower was built in 1889 as the entrance arch for the 1889 World
Fair in Paris.'
print(ask('When was the Eiffel Tower built?', ctx))
print(ask('What was the purpose of the Eiffel Tower?', ctx))
print(ask('What is the capital of France?',ctx))
```

## Przykład 3: Ewaluacja QA na zbiorze SQuAD

```
from datasets import load_dataset
from transformers import pipeline

# Wczytanie 20 przykładów z SQuAD v2
squad = load_dataset('squad_v2', split='validation[:20]')
qa    = pipeline('question-answering',
                 model='deepset/bert-base-cased-squad2')

exact_matches, f1_scores = [], []

def normalize(s):
    return s.lower().strip()

def token_f1(pred, gold):
    p_tok = set(normalize(pred).split())
    g_tok = set(normalize(gold).split())
```

```
    if not p_tok or not g_tok: return 0.0
    common = p_tok & g_tok
    p = len(common) / len(p_tok)
    r = len(common) / len(g_tok)
    return 2 * p * r / (p + r) if (p + r) > 0 else 0.0

for ex in squad:
    if not ex['answers']['text']: continue
    #print(ex['question'],'[',ex['context'],']')
    print(ex['question'])
    print()
    pred  = qa(question=ex['question'], context=ex['context'])['answer']
    print('Answer: ',pred)
    print('-----------------------------------------------------------------')
    gold  = ex['answers']['text'][0]
    em    = int(normalize(pred) == normalize(gold))
    f1    = token_f1(pred, gold)
    exact_matches.append(em)
    f1_scores.append(f1)
print('************************************************************')
print(f'Exact Match: {sum(exact_matches)/len(exact_matches):.3f}')
print(f'Token F1:   {sum(f1_scores)/len(f1_scores):.3f}')
```

## Przykład 4: Prosty RAG z wyszukiwaniem TF-IDF

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
from transformers import pipeline
import numpy as np

# Baza wiedzy (fragmenty dokumentów)
documents = [
    'Python was created by Guido van Rossum in 1991.',
    'The Eiffel Tower is located in Paris and was built in 1889.',
    'Machine learning is a subset of artificial intelligence.',
    'The Amazon River is the largest river by discharge volume.',
    'JavaScript was developed by Brendan Eich in 1995.',
]

vectorizer = TfidfVectorizer()
tfidf_matrix = vectorizer.fit_transform(documents)

def retrieve(query, top_k=2):
    q_vec = vectorizer.transform([query])
    sims  = cosine_similarity(q_vec, tfidf_matrix)[0]
    idx   = np.argsort(sims)[::-1][:top_k]
    return ' '.join([documents[i] for i in idx])
```

```
qa = pipeline('question-answering', model='deepset/bert-base-cased-squad2')

def rag_answer(question):
    context = retrieve(question)
    return qa(question=question, context=context)['answer']

print(rag_answer('Who created Python?'))
print(rag_answer('Where is the Eiffel Tower?'))
print(rag_answer('What is machine learning?'))
```

## 3. Zadania do samodzielnego rozwiązania

**Zadanie 3.1: System QA dla własnych dokumentów**

Wczytaj dowolny artykuł z Wikipedii (np. używając biblioteki wikipedia: pip install wikipedia) na temat wybranego zagadnienia historycznego. Zbuduj system QA, który pozwala zadawać pytania do tego artykułu przy użyciu BERT. Przetestuj minimum 8 różnych pytań (faktograficzne, o daty, osoby, miejsca). Opisz przypadki, gdzie model popełnia błędy.

**Zadanie 3.2: Porównanie BERT vs FLAN-T5 na QA**

Na zbiorze 10 pytań z SQuAD (lub własnych) porównaj wyniki modelu extractive BERT i generatywnego FLAN-T5-base. Oblicz Exact Match i Token F1 dla obu modeli. Znajdź 3 przykłady, gdzie BERT jest lepszy, i 3 przykłady gdzie FLAN-T5 jest lepszy. Wyjaśnij, dlaczego tak jest w każdym przypadku.

**Zadanie 3.3: Rozbudowany RAG z rankingiem**

Rozbuduj system RAG z przykładu 4 o 20 dokumentów (samodzielnie napisanych lub ze Wikipedii). Przetestuj dwa sposoby wyszukiwania: TF-IDF (jak w przykładzie) oraz BM25 (pip install rank-bm25). Porównaj jakość odpowiedzi dla 10 pytań. Zaimplementuj mechanizm wyświetlający top-3 dokumenty z wynikami podobieństwa przed udzieleniem odpowiedzi.

**Zadanie 3.4: Multi-hop Question Answering**

Przygotuj 5 pytań wymagających łączenia informacji z co najmniej 2 dokumentów (np. 'W którym roku urodził się twórca Pythona?' jeśli jeden dok. mówi kto go stworzył, a drugi podaje rok). Zaimplementuj system, który iteracyjnie odpowiada na podpytania i łączy wyniki. Oceń, jak dobrze pojedynczy model BERT radzi sobie z takimi pytaniami.