Практична робота №8

ПОРІВНЯЛЬНИЙ АНАЛІЗ МОДЕЛЕЙ ГЕНЕРАЦІЇ УКРАЇНСЬКОГО ТЕКСТУ (LSTM ТА TRANSFORMER)

Мета роботи: дослідити сучасні підходи до генерації тексту українською мовою. Реалізувати n-грамну модель, рекурентну нейронну мережу LSTM та архітектуру Transformer. Провести порівняльний аналіз якості моделей за допомогою метрик Perplexity та BLEU на базі корпусу українських текстів.

Стек технологій:

Мова програмування: *Python 3.9+*
Бібліотеки: *PyTorch або TensorFlow (Keras), Hugging Face Transformers, NLTK, Scikit-learn.*
Інструменти: *Google Colab або Jupyter Notebook.*

Теоретичний матеріал

Основне завдання мовної моделі - передбачити наступне слово (або символ) на основі попереднього контексту. Математично це виражається як ймовірність послідовності n слів:

$$P(w_1, w_2, \ldots, w_n) = \prod_{i=1}^{n} P(w_i | w_1, \ldots, w_{i-1})$$

У випадку ***n-грам*** (наприклад, триграм), ми припускаємо, що ймовірність слова залежить лише від n-1 попередніх слів (Марковська властивість):

$$P(w_i | w_1, \ldots, w_{i-1}) \approx P(w_i | w_{i-n+1}, \ldots, w_{i-1})$$

LSTM (Long Short-Term Memory)

Це спеціальний вид рекурентної нейронної мережі (RNN), здатної навчатися довготривалим залежностям. На відміну від стандартних RNN, LSTM має "комірку пам’яті" (cell state) та три фільтри (gates):

1. *Forget gate* - що забути з минулого.
2. *Input gate* - яку нову інформацію додати.
3. *Output gate* - що передати на наступний крок.

Transformer

Модель, що повністю відмовилася від рекурентності на користь механізму ***Self-Attention*** (самоуваги). Це дозволяє обробляти всі слова в реченні паралельно, що значно прискорює навчання. Основна формула механізму Attention:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

,

де Q (Query), K (Key), V (Value) - матричні представлення вхідних даних, а d_k - розмірність векторів ключів.

Метрики якості

- *Perplexity (PPL)* - обчислюється як експонента від крос-ентропії. Чим нижче значення PPL, тим впевненіша модель у своїх передбаченнях.
- *BLEU (Bilingual Evaluation Understudy)* - алгоритм оцінки якості тексту, який порівнює згенерований текст із "золотим стандартом" (еталоном) за кількістю спільних n-грам.

Зміст роботи

Завдання 1. *Підготовка даних:*

– *Завантажити тексти (наприклад, твори української літератури або новини).*

– *Виконати токенізацію (розбиття на слова/підслова) та очищення тексту (видалення зайвих символів).*

Методичні рекомендації

Очищення тексту - найважливіший етап. Використовуємо *pymorphy3* для аналізу української морфології.

```
import requests
from bs4 import BeautifulSoup
import re
import nltk
import pandas as pd
import pymorphy3
from collections import Counter
```

Завантаження тексту "Енеїди"

```
url = 'https://www.ukrlib.com.ua/books/getfile.php?tid=1052&type=6'
response = requests.get(url)
```

```
response.encoding = 'utf-8'
```

Парсинг та видалення технічної інформації

```
soup = BeautifulSoup(response.content, 'html.parser')
poem_text = soup.get_text()
poem_text = poem_text.split('КОМЕНТАР ДО', 1)[0] # Відсікаємо примітки
```

Очищення: залишаємо лише українські літери та апостроф

```
poem_text = re.sub(r'[^А-ЩЬЮЯҐЄІЇа-щьюяґєіїʼ\s]', ' ', poem_text)
words = [word.lower() for word in poem_text.split() if len(word) > 1]
```

Робота зі стоп-словами та лематизація

```
morph = pymorphy3.MorphAnalyzer(lang='uk')
stop_words = set("з, у, як, що, тут, щоб, так, то, по, та, а, або, б, в,
від, він, вона, вони, все, всі, до, за, і, із, й, на, не, ні, про,
собі".split(", "))

clean_words = [morph.parse(w)[0].normal_form for w in words if w not in
stop_words]

print(f"Кількість слів після очищення: {len(clean_words)}")
print(f"Топ-5 слів: {Counter(clean_words).most_common(5)}")
```

Завдання 2. *Реалізація моделей:*

– *Побудувати Baseline (n-грамну модель) для порівняння.*

– *Спроектувати та навчити LSTM мережу на рівні символів або слів.*

– *Використати архітектуру Transformer (наприклад, GPT-2 small або налаштувати власну невелику модель).*

Методичні рекомендації

N-gram модель (Baseline) – модель, яка будує словник імовірностей наступним чином: яке слово найімовірніше йде після заданої пари слів.

```
from collections import defaultdict

def build_ngram_model(words, n=3):
    model = defaultdict(Counter)
    for i in range(len(words) - n):
        context = tuple(words[i:i+n-1])
        target = words[i+n-1]
        model[context][target] += 1
    return model

ngram_model = build_ngram_model(words, n=3)
```

LSTM модель (Рівень слів)

```
import torch
import torch.nn as nn

class LSTMModel(nn.Module):
    def __init__(self, vocab_size, embed_size=128, hidden_size=256):
        super(LSTMModel, self).__init__()
        self.embedding = nn.Embedding(vocab_size, embed_size)
        self.lstm = nn.LSTM(embed_size, hidden_size, batch_first=True)
        self.linear = nn.Linear(hidden_size, vocab_size)

    def forward(self, x, h):
        x = self.embedding(x)
        out, h = self.lstm(x, h)
        out = self.linear(out[:, -1, :]) #Прогноз останнього слова в
послідовності
        return out, h
```

Словники для перетворення текст <-> числа

```
unique_words = sorted(list(set(words)))
word2idx = {word: i for i, word in enumerate(unique_words)}
idx2word = {i: word for i, word in enumerate(unique_words)}
```

Визначаємо параметри моделі

```
vocab_size = len(unique_words)
embed_size = 128
hidden_size = 256
```

Створюємо екземпляр моделі (Ініціалізація)

```
model_lstm = LSTMModel(vocab_size, embed_size, hidden_size)
```

Якщо ви просто створили модель і відразу запустили генерацію без навчання (training), результат буде набором випадкових слів. Для отримання осмисленого тексту модель має пройти цикл навчання:

- Пройти по тексту words.
- Порахувати помилку (Loss).
- Оновити ваги (Optimizer.step()).

Реалізація Transformer

Найпростіший шлях - використати архітектуру GPT-2 через бібліотеку Hugging Face. Для української мови найкраще підійдуть моделі ukrainian-gpt2. Якщо будуєте власну архітектуру, зверніть увагу на Positional Encoding, оскільки трансформер не знає порядку слів без них.

```
input_ids = tokenizer.encode(prompt, return_tensors='pt')
output = transformer_model.generate(input_ids, max_length=60, num_beams=5,
                                    no_repeat_ngram_size=2,
early_stopping=True)
gen_trans = tokenizer.decode(output[0], skip_special_tokens=True)

print(f"Transformer Output\n{gen_trans}")
```

Завдання 3. *Виконати генерацію тексту:*

– *Задати однаковий prompt для всіх моделей.*

– *Згенерувати продовження тексту (50–100 слів).*

Методичні рекомендації

```
prompt = "еней був хлопець"
```

N-gram генерація

```
def gen_ngram(model, start_text, length=30):
    context = tuple(start_text.lower().split()[-2:])
    res = list(context)
    for _ in range(length):
        if context in model:
            next_word = random.choices(list(model[context].keys()),
                                       weights=model[context].values())[0]
            res.append(next_word)
            context = tuple(res[-2:])
        else: break
    return " ".join(res)

print(f"N-gram Output :\n{gen_ngram(ngram_model, prompt)}")
```

Transformer генерація

```
input_ids = tokenizer.encode(prompt, return_tensors='pt')
output = transformer_model.generate(input_ids, max_length=60, num_beams=5,
                                    no_repeat_ngram_size=2,
early_stopping=True)
gen_trans = tokenizer.decode(output[0], skip_special_tokens=True)

print(f"Transformer Output:\n{gen_trans}")
```

Завдання 4. *Провести оцінку отриманих результатів:*

– *Розрахувати Perplexity для тестової вибірки.*

– *Порівняти згенеровані тексти за допомогою метрики BLEU щодо контрольних речень.*

Порівняльна таблиця результатів:

Модель	Perplexity	BLEU Score	Швидкість генерації
N-gram			
LSTM			
Transformer			

Методичні рекомендації

Оцінюємо моделі за двома критеріями: математична точність (Perplexity) та лінгвістична схожість (BLEU).

Як інтерпретувати результати:

- Perplexity < 50 - модель добре вивчила структуру мови.
- Perplexity > 150 - модель "розгублена", текст буде хаотичним.
- BLEU > 0.4 - текст дуже близький до людського оригіналу (для специфічних завдань).
- BLEU 0.1 - 0.2 - типовий результат для вільної генерації.

```
from nltk.translate.bleu_score import sentence_bleu, SmoothingFunction
import numpy as np
```

BLEU Score (порівняння з оригінальним уривком)

```
reference = [words[10:40]] # Еталонний текст з книги
candidate = gen_trans.split()
smooth = SmoothingFunction().method1
score = sentence_bleu(reference, candidate, smoothing_function=smooth)
```

Розрахунок Perplexity (на прикладі функції втрат Cross-Entropy)

```
# PPL = exp(loss)
loss_example = 3.5
ppl = np.exp(loss_example)

print(f"Метрики для Transformer:")
print(f"BLEU Score: {score:.4f}")
print(f"Perplexity: {ppl:.2f}")
```

Завдання 5. *Індивідуальні завдання (15 варіантів)*

1	**Поетична генерація**	Навчити LSTM на віршах Т. Шевченка та порівняти ритміку з n-грамною моделлю
2	**Юридичний стиль**	Дослідити якість генерації офіційно-ділових текстів (договорів) за допомогою Transformer
3	**Діалектна**	Порівняти роботу моделей на літературній мові та суржику

	адаптація	(використовуючи специфічні датасети)
4	**Аналіз температурного режиму**	Провести експеримент, як зміна параметру temperature від 0.1 до 2.0 впливає на Perplexity.
5	**Вплив токенізації**	Порівняти Byte Pair Encoding (BPE) та WordPiece токенізацію для Transformer в контексті української морфології
6	**Генерація заголовків**	Навчити моделі створювати клікбейтні заголовки для українських новинних ресурсів
7	**Детекція плагіату**	Реалізувати механізм, який порівнює, наскільки згенерований текст відрізняється від навчального корпусу (overfitting check).
8	**Стилістичне перенесення**	Спробувати згенерувати технічну інструкцію у стилі Лесі Українки (Fine-tuning Transformer)
9	**Оптимізація LSTM**	Дослідити вплив кількості шарів (Stacking LSTM) на логічну зв'язність тексту
10	**Bidirectional LSTM**	Порівняти односпрямовану та двоспрямовану LSTM для завдання заповнення пропущених слів у реченні
11	**Стиснення моделей**	Застосувати квантизацію (quantization) до моделі Transformer та виміряти втрату якості за метрикою BLEU
12	**Генерація програмного коду**	Навчити n-грамну модель передбачати наступні токени в коді на Python з українськими коментарями
13	**Аналіз помилок**	Створити класифікатор типових помилок (граматичних, логічних), які робить LSTM у порівнянні з Transformer
14	**Малоресурсне навчання**	Навчити Transformer на дуже малому обсязі даних (до 1 Мб) і проаналізувати момент початку галюцинацій
15	**Інтеграція словників**	Спробувати інтегрувати зовнішній словник синонімів у процес генерації n-грамної моделі для покращення BLEU

Контрольні запитання

1. Що таке мовна модель?
2. У чому полягає "прокляття розмірності" в n-грамних моделях при збільшенні n?
3. Як працює LSTM?
4. Чому LSTM краще справляється з контекстом, ніж звичайна RNN? Поясніть роль "state cell".
5. У чому перевага Transformer?

6. Поясніть різницю між Self-Attention та Cross-Attention.
7. Навіщо у формулі Attention використовується масштабування?
8. Як метрика Perplexity пов'язана з крос-ентропією?
9. Які недоліки має метрика BLEU при оцінці творчих (художніх) текстів?
10. Що таке "температура" (temperature) при генерації тексту і як вона впливає на різноманітність слів?
11. Порівняйте обчислювальну складність навчання LSTM та Transformer для довгих послідовностей.