НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені Ігоря Сікорського»
Факультет Інформатики та обчислювальної техніки
Кафедра Інформаційних систем та технологій

ЛАБОРАТОРНА РОБОТА №6

з дисципліни «Обробка та аналіз текстових даних на Python»
на тему «Класифікація тексту з використанням нейронних мереж»
Варіант №2

Виконала: студентка групи ІС-з21
Коломієць Катерина Миколаївна
28.05.2025

Перевірив: асист. Мягкий М. Ю.

Київ – 2025

GitHub: https://github.com/kate-miiets-uni/oatd-py

Теоретичні відомості

LSTM (Довга Короткострокова Пам'ять) – це особливий, розширений тип рекурентних нейронних мереж (RNN), розроблений для ефективної роботи з послідовними даними (такими як текст, аудіо, часові ряди). Їхня головна перевага полягає у здатності вивчати та запам'ятовувати довгострокові залежності, що є серйозною проблемою для традиційних RNN.

На кожному часовому кроці, LSTM-комірка виконує наступні дії:

1. Приймає входи: поточний вхід і прихований стан з попереднього кроку.

2. Обчислює ворота забування: вирішує, яку частину старої пам'яті слід викинути.

3. Обчислює ворота входу: вирішує, яка нова інформація має бути додана до пам'яті.

4. Оновлює стан комірки пам'яті: комбінує стару пам'ять (після забування) з новою релевантною інформацією. Це "додавання" і "видалення" інформації з комірки пам'яті є ключовим, оскільки воно дозволяє градієнтам текти більш стабільно і зберігати інформацію на довгі дистанції.

5. Обчислює ворота виходу: вирішує, яку частину оновленої пам'яті слід вивести як новий прихований стан.

Хід роботи

1. Імпортуємо бібліотеки:

- requests – для роботи з HTTP-запитами;

- re – для роботи з регулярними виразами;

- numpy – для роботи з багатовимірними масивами та математичними операціями над ними;

- bs4 – для якісного витягування тексту з веб-сторінок;

- tensorflow.keras – для попередньої обробки тексту, що дозволяє розбивати текст на токени та перетворювати їх на числові послідовності;

- sklearn – для розділення наборів даних на тренувальні та тестові підмножини.

```
import requests
import re
import numpy as np
from bs4 import BeautifulSoup
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from sklearn.model_selection import train_test_split
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense
from sklearn.metrics import precision_score, recall_score
```

2. Імпортуємо 10 новин: 5 спортивних, 5 культурних.

```
# Список новин (5 спортивних, 5 культурних)
urls = [
    'https://www.bbc.com/ukrainian/articles/c1k4x2xp124o',
    'https://www.bbc.com/ukrainian/articles/ckgyqkrn5n6o',
    'https://www.bbc.com/ukrainian/articles/crlrkpk0ywro',
    'https://www.bbc.com/ukrainian/articles/c3w66qz4j8yo',
    'https://www.bbc.com/ukrainian/articles/cl7yyw8xwg9o',
    'https://www.bbc.com/ukrainian/articles/cqj77vv07x2o',
    'https://www.bbc.com/ukrainian/articles/c4grdnxg0l6o',
    'https://www.bbc.com/ukrainian/articles/cwy630j7ed1o',
    'https://www.bbc.com/ukrainian/articles/cp8kyze7zm3o',
    'https://www.bbc.com/ukrainian/articles/cy9derrnlzzo',
]
```

Для кожної новини заносимо її у відповідний лейбл (потрібно для подальшого навчання мережі), де 0 – спортивні новини, 1 – культурні. Перевіряємо на позитивну відповідь на запит про отримання html-сторінки, очищуємо сторінку від тегів, дістаємо сам текст з тегів <p>.

```
documents = []                                                    6 9 8
# Список для зберігання міток класів (0 для спорту, 1 для культури)
labels = []

# Для кожного посилання зі списку посилань
for i, url in enumerate(urls):
    # Визначаємо мітку класу: перші 5 - спорт (0), наступні 5 - культура (1)
    label = 0 if i < 5 else 1
    labels.append(label)

    # Надсилаємо запит на сервер про отримання тексту новини
    response = requests.get(url)
    # Якщо запит вдалий
    if response.status_code == 200:
        # Видаляємо з тексту новини усі теги
        soup = BeautifulSoup(response.text, features: 'html.parser')
        # Видобуваємо текст з тегів <p>
        text = ' '.join([p.text for p in soup.find_all('p')])  # Отримуємо текст із теґі
        # Додаємо текст у список з документами
        documents.append(text)

print(documents[0][:100])
```

Приклад отриманого тексту:

```
Автор фото, Getty Images Український боксер, чемпіон світу в надважкій вазі за версіями WBO, WBA, WB
```

3. Очищуємо текст від стоп-слів, коротких слів (менше 3 символів), від будь-яких символів, що не літери, цифри або _, а також зводимо слова до нижнього регістру для уніфікації слів. Повертаємо текст кожної новини у вигляді рядка.

```python
# Функція приводить текст до нижнього регістру, очищає його від пунктуації та стоп-сл
def clean_text(text):  1 usage
    # Сет стоп-слів
    stop_words = {"щоб", "про", "але", "для", "від", "через", "також", "якщо", "вже",
                  "його", "може", "який", "яка", "які", }
    # Приводимо до нижнього регістру для уніфікації слів
    text = text.lower()
    # Видаляємо усі символи окрім літер, цифр та _
    text = re.sub( pattern: r'\W+', repl: ' ', text)
    # Розділяємо текст на слова для видалення стоп-слів
    words = text.split()
    # Видаляємо стоп-слова та короткі слова (менше 3-х символів)
    words = [word for word in words if word not in stop_words and len(word) > 2]
    return ' '.join(words)
```

Приклад отриманого очищеного тексту:

```
автор фото getty images український боксер чемпіон світу надважкій вазі версіями wbo wba wbc ibo річ
```

4. Підготовлюємо тексти новин для створення мережі.

Токенізуємо слова за допомогою Tokenizer(), який допомагає побудувати словник зі всіх унікальних слів у корпусі та присвоїти кожному слову унікальний числовий ідентифікатор. num_words=5000 означає, що зберігатимуться лише 5000 найчастіше вживаних слів з усього корпусу документів. oov_token="<unk>" означає, що слова, які не увійдуть до 5000 найчастіших даних, будуть мати токен <unk>.

.fit_on_texts() для навчання токенізатора. Метод проходиться по всіх очищених текстах та збирає всі унікальні слова, на основі яких створює словник.

.texts_to_sequences() замінює усі слова у тексті на відповідні їм ідентифікатори зі словника, слова, які не потрапили до словника отримують ідентифікатор <unk>.

```python
# Токенізація текстів
tokenizer = Tokenizer(num_words=5000, oov_token="<unk>")
tokenizer.fit_on_texts(cleaned_documents)
sequences = tokenizer.texts_to_sequences(cleaned_documents)
```

Вирівнюємо довжини новин. Оскільки тексти новин вже представлені списками слів, то ми можемо вирівняти кожен список за кількістю чисел. Це виконувати потрібно, адже нейронна мережа LSTM працює з фіксованими входами, тобто розмір вхідних даних має бути однаковим. У нашому випадку ми скорочуємо текст новини, якщо він перевищує певну кількість чисел (слів) у списку. padding='post' означає, що 0, які будуть додані до списку чисел новини, довжина якої менша за фіксовану, будуть додані в кінець списку. truncating='post' означає, що «обрізка», яку здійснять до списку, чия довжина певищує максимально задану, буде виконана на кінці списку.

```
# Padding (вирівнювання довжини послідовностей)
max_len = max([len(x) for x in sequences]) if sequences else 0
if max_len < 150:
    max_len = 150

padded_sequences = pad_sequences(sequences, maxlen=max_len, padding='post', truncating='pos
```

```
# Padding (вирівнювання довжини послідовностей)
max_len = max([len(x) for x in sequences]) if sequences else 0
if max_len < 100:
    max_len = 100

padded_sequences = pad_sequences(sequences, maxlen=max_len, padding='post', truncating='post
```

Перетворюємо список міток на масив типу NumPy для легшої роботи з TensorFlow.

```
# Перетворюємо мітки на масив NumPy
labels = np.array(labels)
```

Виводимо інформативні дані після підготовки даних:

```
print(f"\nКількість унікальних токенів у словнику: {len(tokenizer.word_index)}")
print(f"Максимальна довжина послідовності: {max_len}")
print(f"Форма підготовлених даних (padded_sequences): {padded_sequences.shape}")
print(f"Форма міток (labels): {labels.shape}")
```

```
Кількість унікальних токенів у словнику: 3942
Максимальна довжина послідовності: 1231
Форма підготовлених даних (padded_sequences): (10, 1231)
Форма міток (labels): (10,)
```

5. Ділимо дані на тренувальний та тестові набори. У нас padded_sequences виступатимуть у ролі матриці, labels – у ролі вектора, test_size=0.2 визначає яка частка даних піде в тестовий набір, random_state=42 забезпечує відтворюваність результату при кожному новому запуску програми, stratify=labels забезпечує пропорційність розподілу даних у тренувальні набори та в тестові.

Виводимо результати розділення.

```
# Розділення даних на тренувальний та тестовий набори
# 80% для навчання, 20% для тестування
X_train, X_test, y_train, y_test = train_test_split(padded_sequences, labels, test_size=0.2, random_state=42, stratify=labels)

print(f"\nРозмір тренувального набору: {len(X_train)} документів")
print(f"Розмір тестового набору: {len(X_test)} документів")
```

```
Розмір тренувального набору: 8 документів
Розмір тестового набору: 2 документів
```

6. Визначаємо ключові парамери:

vocab_size = len(tokenizer.word_index) + 1 визначає розмір словника моделі + враховує наявність індексу для слів <unk>.

embedding_dim = 100 визначає розмірність вектора вбудовування (embedding vector) для кожного слова для кращого сементатичного розпізнавання моделлю зв’язків між словами.

lstm_units = 128 визначає кількість одиниць (нейронів) у шарі LSTM. Більша кількість одиниць дозволяє моделі запам'ятовувати складніші патерни та довші залежності, але також збільшує обчислювальні витрати та ризик перенавчання.

```
# Побудова та навчання LSTM-моделі
vocab_size = len(tokenizer.word_index) + 1 # Розмір словника (+1 для <unk> токена)
embedding_dim = 100 # Розмірність векторів вбудовування слів
lstm_units = 128 # Кількість одиниць у шарі LSTM
```

7. Будуємо модель за допомогою Sequential([...]), яка створює послідовну модель Keras, що означає, що шари будуть додаватися один за одним у лінійній послідовності.

Embedding() - перший шар у мережі, відповідає за перетворення числових індексів слів на щільні вектори вбудовування. input_dim=vocab_size вказує, скільки унікальних індексів слів може бути на вході, output_dim=embedding_dim визначає довжину вектора вбудовування для кожного слова, input_length=max_len вказує фіксовану довжину вхідних послідовностей, яку ви встановили за допомогою padding.

LSTM() - основний шар моделі для обробки тексту. Він має внутрішні "ворота" (gates), які дозволяють йому контролювати потік інформації та вирішувати, яку інформацію запам'ятовувати, яку забувати, а яку передавати далі. Це дозволяє LSTM захоплювати довгострокові залежності в тексті – тобто, зв'язки між словами, які знаходяться далеко одне від одного в реченні чи документі. На вхід LSTM отримує послідовність векторів з шару Embedding, обробляє їх і видає один вихідний вектор, який підсумовує інформацію з усієї послідовності.

Dense() - вихідний шар моделі. units=1 - оскільки це завдання бінарної класифікації (спорт або культура), нам потрібен лише один вихідний нейрон, activation='sigmoid' - сигмоїдна функція активації, яка стискає вивід цього нейрона до діапазону від 0 до 1.

```
model = Sequential([
    # Шари моделі
    # Embedding Layer: Перетворює індекси слів у щільні вектори
    Embedding(input_dim=vocab_size, output_dim=embedding_dim),
    # LSTM Layer: Обробляє послідовності, захоплюючи залежності на великих відстанях
    LSTM(units=lstm_units),
    # Dense Output Layer: Бінарна класифікація з сигмоїдною активацією
    Dense(units=1, activation='sigmoid')
])
```

8. Компілюємо модель, визначаємо як вона буде навчатися. optimizer='adam' – оптимізатор, коригує ваги нейронів у мережі таким чином, щоб модель ставала кращою у виконанні свого завдання, loss='binary_crossentropy' - функція втрат, також відома як функція помилки або функція вартості, вимірює, наскільки "погано" модель працює, metrics=['accuracy'] використовуються для моніторингу ефективності моделі під час навчання та після нього.

```
# Компіляція моделі
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

9. Запускаємо процес навчання моделі, де модель пройде по тренувальних даних 10 разів, оновлення ваг моделі відбувається після обробки 32 зразків, 20% тренувальних даних використовується для валідації на кожній епосі.

```
print("\n--- Процес навчання моделі ---")
history = model.fit(X_train, y_train, epochs=10, batch_size=32, validation_split=0.2, v
```

```
--- Процес навчання моделі ---
Epoch 1/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 2s 2s/step - accuracy: 0.6667 - loss: 0.6922 - val_accuracy: 0.5000 - val_loss: 0.6937
Epoch 2/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 323ms/step - accuracy: 0.6667 - loss: 0.6857 - val_accuracy: 0.5000 - val_loss: 0.6950
Epoch 3/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 290ms/step - accuracy: 0.6667 - loss: 0.6795 - val_accuracy: 0.5000 - val_loss: 0.6976
Epoch 4/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 283ms/step - accuracy: 0.6667 - loss: 0.6732 - val_accuracy: 0.5000 - val_loss: 0.7021
Epoch 5/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 280ms/step - accuracy: 0.6667 - loss: 0.6667 - val_accuracy: 0.5000 - val_loss: 0.7104
Epoch 6/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 289ms/step - accuracy: 0.6667 - loss: 0.6604 - val_accuracy: 0.5000 - val_loss: 0.7255
Epoch 7/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 312ms/step - accuracy: 0.6667 - loss: 0.6552 - val_accuracy: 0.5000 - val_loss: 0.7444
Epoch 8/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 280ms/step - accuracy: 0.6667 - loss: 0.6519 - val_accuracy: 0.5000 - val_loss: 0.7441
Epoch 9/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 268ms/step - accuracy: 0.6667 - loss: 0.6449 - val_accuracy: 0.5000 - val_loss: 0.7344
Epoch 10/10
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 305ms/step - accuracy: 0.6667 - loss: 0.6358 - val_accuracy: 0.5000 - val_loss: 0.7263
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 323ms/step
```

10. Оцінюємо якість класифікації за допомогою метрик precision та recall.

```
#  Оцінка моделі на тестовому наборі
y_pred_proba = model.predict(X_test) # Отримуємо ймовірності належності до позитивного класу
y_pred = (y_pred_proba > 0.5).astype(int) # Перетворюємо ймовірності на бінарні класи (0 або 1)

# Обчислюємо precision (точність) та recall (повноту)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)

print(f"\n--- Результати класифікації ---")
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
```

```
--- Результати класифікації ---
Precision: 0.5000
Recall: 1.0000
```

Результати.

З виводу процесу навчання нейронної мережі можемо зробити такі висновки:

- тренувальна точність (accuracy) стабільна, залишається на рівні 0.6667 протягом усіх 10 епох;

- тренувальні втрати (loss) зменшуються, вони повільно спадають від 0.6922 до 0.6358, це означає, що модель вчиться і стає кращою на даних, які вона бачить;

- валідаційна точність (val_accuracy) стабільна, але низька: залишається на рівні 0.5000 протягом усіх 10 епох;

- валідаційні втрати (val_loss) збільшуються, вони зростають від 0.6937 до 0.7263, що означає, що відбувається перенавчання (overfitting).

Результати за метриками: модель правильно класифікує тільки 50% наборів, при цьому вона визначає культурні новини як культурні у 100% випадків. Тобто, можемо припускати, що наша модель має певне «упередження» до прогнозування культурних новин, при цьому вона позначає спортивні новини як культурні також.

Отже, розроблена LSTM-модель успішно запускається і починає навчання, але поточні результати чітко вказують на проблему перенавчання (overfitting), модель запам'ятовує тренувальні дані (що видно по зменшенню тренувальних втрат), але не може узагальнювати на нові, невидимі приклади (про це свідчить зростання валідаційних втрат та низька валідаційна точність). Головна причина такої поведінки — критично малий обсяг даних. Для ефективного навчання нейронних мереж, особливо таких як LSTM, потрібні сотні, а краще тисячі або десятки тисяч прикладів.

Висновок

Отже, у ході лабораторної роботи ми вивчили LSTM-моделі, як вони утворюються, а також імпортували 10 новин, виконали їхнє очищення та підготовлення для використання у

будуванні нейронної мережі. Розділили дані, побудували модель, навчили та зкомпілювали її, а також провели аналіз вихідних даних та підбили такі підсумки: модель вдало навчалася, проте через низьку кількість матеріалів для навчання, вона не змогла якісно класифікувати нові для себе дані.

Контрольні запитання

1. Що таке precision і recall?

Precision - вимірює якість позитивних прогнозів моделі. Вона показує, наскільки модель є "точною" або "влучною" у своїх позитивних твердженнях. Precision відповідає на питання: "З усіх елементів, які модель спрогнозувала як позитивні, скільки з них були насправді позитивними?"

Recall - вимірює кількість або повноту виявлення всіх справжніх позитивних випадків. Вона показує, наскільки добре модель "згадує" або "витягує" всі релевантні елементи. Recall відповідає на питання: "З усіх елементів, які насправді були позитивними, скільки з них модель правильно ідентифікувала?"

2. Як розраховуються precision та recall у класифікації?

Precision: TP/(TP+FP)

Recall: TP/(TP+FN)

Причому, True Positives (TP - Істинно Позитивні): модель правильно спрогнозувала позитивний клас; False Positives (FP - Хибно Позитивні): модель неправильно спрогнозувала позитивний клас; False Negatives (FN - Хибно Негативні): модель неправильно спрогнозувала негативний клас, хоча насправді це був позитивний клас.

3. Чим precision відрізняється від accuracy?

Accuracy вимірює загальну частку правильних прогнозів моделі з усіх зроблених прогнозів. Вона відповідає на питання: "Як часто модель загалом робить правильні передбачення?".

Precision вимірює частку правильних позитивних прогнозів серед усіх прогнозів, які модель позначила як позитивні. Вона відповідає на питання: "Коли модель каже, що це позитивний клас, наскільки вона права?".

4. Яка роль LSTM у NLP?

LSTM (Long Short-Term Memory) нейронні мережі відіграють вирішальну роль в обробці природної мови (NLP) завдяки своїй унікальній здатності ефективно працювати з послідовними даними, якими є текст. Завдяки цій здатності зберігати інформацію протягом тривалого часу, LSTM є незамінними для широкого спектру завдань NLP, включаючи класифікацію тексту (як у вашому прикладі), машинний переклад, генерацію тексту, аналіз

тональності та розпізнавання мови, забезпечуючи глибоке розуміння та обробку мовних даних.

5. Як працює padding при роботі з текстами у нейромережах?

При роботі з текстом у нейронних мережах, таких як LSTM, критично важливо, щоб усі вхідні дані мали однакову довжину, хоча насправді речення та документи різняться за розміром. Для вирішення цієї проблеми застосовується padding (вирівнювання): якщо послідовність слів (після їх перетворення на числа) коротша за визначену максимальну довжину (max_len), вона доповнюється нулями до цієї довжини (зазвичай в кінці); якщо ж послідовність довша, вона обрізається до max_len. Це забезпечує уніфікований формат вхідних даних, дозволяючи нейронній мережі ефективно обробляти текстові дані пачками, що є ключовим для прискорення навчання та коректної роботи моделі.

6. Як розбити текст на токени для навчання моделі?

Розбити текст на токени можна за допомогою Keras Tokenizer, який автоматизує процес розбиття тексту на токени (зазвичай слова) та їхнє перетворення на числові послідовності.

7. Що таке dropout і як він допомагає уникнути перенавчання?

Dropout – це техніка регуляризації, що використовується в нейронних мережах для боротьби з перенавчанням (overfitting). Під час навчання моделі, шар Dropout випадковим чином "вимикає" (тобто, тимчасово ігнорує або встановлює їхній вихід на нуль) певну частку нейронів у попередньому шарі. Ця частка визначається як коефіцієнт відсіву (dropout rate), наприклад, Dropout(0.5) означає, що 50% нейронів будуть випадково вимкнені в кожній ітерації. Кожен раз, коли мережа бачить новий "батч" даних під час навчання, вибирається інший випадковий набір нейронів для вимкнення.

8. Чому важливо нормалізувати тексти перед навчанням моделі?

Нормалізація тексту перед навчанням моделі є критично важливим етапом попередньої обробки в NLP, оскільки вона забезпечує послідовність і зменшує шум у даних, що подаються до моделі. Це допомагає моделі краще вивчати значущі закономірності, а не відволікатися на незначні варіації.

9. Як можна оптимізувати LSTM-архітектуру?

Оптимізувати модель можна налаштуванням гіперпараметрів, додаванням dropout, використанням попередньо вже навчених вбудувань, зміною самої архітектури моделі (наприклад, додаванням шарів), можна вносити зміни в налаштування процесу навчання, змінювати оптимізатор, швидкість навчання, ставити ранню зупинку.

10. Як візуалізувати результати класифікації?

Можна використати матрицю плутанини, ROC-криву, яка відображає співвідношення між True Positive Rate (TPR) (той самий Recall), і False Positive Rate (FPR) – часткою хибно позитивних спрацьовувань серед усіх справжніх негативних випадків.