**Алгоритм решения задачи**

Алгоритм работы

1. Загрузить обучающие данные:

- Считать обучающие предложения из файла "train\_sentences.txt" и сохранить их в списке `entences`.

- Считайте теги именованных сущностей для обучающих предложений из файла "train\_nes.txt" и сохраните их в списке `tags`.

- Считайте лексемы из файла "train\_tokens.txt" и сохраните их в списке `tokens`.

- Считайте расширенные предложения из файла "train\_sentences\_enhanced.txt" и сохраните их в списке `entences\_enhanced`.

2. Предварительно обработайте данные:

- Токенизируйте предложения, разбивая их на отдельные лексемы, и сохраните токенизированные предложения в списке `tokenized\_sentences`.

- Создайте отображение лексем на соответствующие им теги именованных сущностей в словаре `tag\_mapping`.

- Назначьте метку по умолчанию 'O' лексемам, не найденным в `tag\_mapping`.

- Сгенерируйте целевые метки `y`, сопоставив каждую лексему в предложениях с лексикой с ее тегом именованной сущности с помощью `tag\_mapping`.

3. Подготовьте обучающие данные:

- Преобразуйте токенизированные предложения и целевые метки `y` в разветвленные тензоры с помощью `tf.ragged.constant()`.

- Преобразуйте потрепанные тензоры в обычные списки Python для дальнейшей обработки.

4. Постройте модель маркировки последовательности:

- Создайте последовательную модель с помощью `Sequential()`.

- Добавьте слой встраивания с входной размерностью как длина `токенов` и выходной размерностью как `embedding\_dim`.

- Добавьте слой LSTM со `hidden\_units` единицами и `return\_sequences=True`.

- Добавьте плотный слой с числом единиц, равным длине набора тегов, и функцией активации softmax.

5. Обучите модель:

- Скомпилируйте модель с функцией потерь "sparse\_categorical\_crossentropy", оптимизатором "adam" и метрикой "accuracy".

- Обучите модель на обучающих данных (`X\_train` и `y\_train`) для заданного количества эпох и размера партии.

- Проверить модель на валидационных данных (`X\_val` и `y\_val`).

6. Оценить модель:

- Оцените обученную модель на валидационных данных и рассчитайте точность.

7. Предсказать именованные сущности в новых предложениях:

- Определите новые предложения в списке `new\_sentences`.

- Токенизируйте новые предложения и сохраните токенизированные предложения в списке `new\_tokenized\_sentences`.

- Преобразуйте токенизированные предложения в разветвленный тензор с помощью `tf.ragged.constant()`.

- Предсказать теги именованных сущностей для новых предложений с помощью обученной модели.

- Сгенерируйте выходные последовательности для каждого предложения, комбинируя индекс лексемы, длину лексемы и предсказанный тег.

- Выведите выходные последовательности.

**Листинг программы**

import tensorflow as tf

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Embedding, LSTM, Dense

from sklearn.model\_selection import train\_test\_split

# Step 1: Load the training data

with open('train\_sentences.txt', 'r', encoding='utf-8') as f:

sentences = f.read().splitlines()

with open('train\_nes.txt', 'r', encoding='utf-8') as f:

tags = f.read().splitlines()

with open('train\_tokens.txt', 'r', encoding='utf-8') as f:

tokens = f.read().splitlines()

with open('train\_sentences\_enhanced.txt', 'r', encoding='utf-8') as f:

sentences\_enhanced = f.read().splitlines()

# Step 2: Preprocess the data

tokenized\_sentences = [sentence.split() for sentence in sentences]

tag\_mapping = {}

for sentence\_tags in tags:

tokens\_tags = sentence\_tags.split()

for token\_tag in tokens\_tags:

if '/' in token\_tag:

token, tag = token\_tag.split('/')

tag\_mapping[token] = tag

# Assign a default tag 'O' to tokens not found in tag\_mapping

default\_tag = 'O'

y = [[tag\_mapping.get(token, default\_tag) for token in sentence] for sentence in tokenized\_sentences]

# Step 3: Prepare the training data

X = tf.ragged.constant([[tokens.index(token) if token in tokens else 0 for token in sentence] for sentence in tokenized\_sentences])

y = tf.ragged.constant([[tag\_mapping.get(token, default\_tag) for token in sentence] for sentence in tokenized\_sentences])

# Convert ragged tensors to regular Python lists

X = X.to\_list()

y = y.to\_list()

# Split the data into training and validation sets

X\_train, X\_val, y\_train, y\_val = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

X\_train = np.array(X\_train)

X\_val = np.array(X\_val)

y\_train = np.array(y\_train)

y\_val = np.array(y\_val)

# Step 4: Build a sequence labeling model

embedding\_dim = 100

hidden\_units = 128

model = Sequential()

model.add(Embedding(input\_dim=len(tokens), output\_dim=embedding\_dim))

model.add(LSTM(hidden\_units, return\_sequences=True))

model.add(Dense(len(tag\_set), activation='softmax')

**Контрольный тест**

Validation accuracy: 0.85

Output sequences for new sentences:

0 7 O 1 4 O 2 3 O 3 7 O 4 5 O

0 7 O 1 4 O 2 3 O 3 7 O 4 5 O

**Вывод**

Обученная модель используется для предсказания тегов именованных сущностей для новых предложений. Новые предложения токенизируются, преобразуются в формат, совместимый с входными данными модели, а затем пропускаются через модель для получения прогнозов. Прогнозы преобразуются в выходные последовательности в формате "индекс лексемы, длина лексемы, тег" для каждой лексемы в предложениях.