УДК 519.68:007.5

Лабораторная работа 3. Рекуррентные сети

Цели работы: построение рекуррентных сетей на базе LSTM для решения задач классификации текстов и регрессионного анализа последовательных данных.

Ключевые слова: сети долгой кратковременной памяти, анализ временных рядов, кодирование естественного языка.

§ 1. Классификация текстов

1.1. Подготовка к работе. Введем следующие команды в рабочей тетради jupyter и проверим, что они исполнились без ошибок:

```
import os
from operator import itemgetter
import numpy as np
import pandas as pd

import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
get_ipython().magic(u'matplotlib inline')
plt.style.use('ggplot')

import warnings
warnings.filterwarnings('ignore')

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import models, regularizers, layers,
                             optimizers, losses, metrics
from keras.models import Sequential
from keras.layers import Dense, LSTM, Embedding, Bidirectional
from keras.utils import to_categorical
from keras.datasets import reuters
print(os.getcwd())
print("Modules imported \n")
```

Загружаем базу текстов reuters, разбитых на 46 классов в зависимости от их тематики.

```
Num_words=10000 # ограничимся текстами с таком max длиной
(train_data, train_labels), (test_data, test_labels) =
                             reuters.load_data(num_words=Num_words )
```

Проверим какие размерности у загруженной базы текстов и меток.

```
print("train_data ", train_data.shape)
print("train_labels ", train_labels.shape)

print("test_data ", test_data.shape)
print("test_labels ", test_labels.shape)
```

Преобразуем целочисленные коды train_data с помощью словаря из датасета обратно в текст и распечатаем его.

```
word_index = reuters.get_word_index()
reverse_word_index =
    dict([(value, key) for (key, value) in word_index.items()])
decoded_newswire = ' '.join([reverse_word_index.get(i - 3, '?') for i in
                                                    train_data[0]])

print(decoded_newswire)
print(train_labels[0])
```

Подготовим примитивную функцию для векторизации текстовых кодов.

```
def vectorize_sequences(sequences, dimension=Num_words):
    results = np.zeros((len(sequences), dimension))
    for i, sequence in enumerate(sequences):
        results[i, sequence] = 1.
    return results
```

Для обучения нашей модели ограничимся первыми 11 классами текстов.

```
indx_train = train_labels <= 10
indx_test = test_labels   <= 10

train_data  = train_data[indx_train]
train_labels  = train_labels[indx_train]

test_data  = test_data[indx_test]
test_labels    = test_labels  [indx_test]
```

Проведем векторизацию данных для обучения и тестированя.

```
x_train = vectorize_sequences(train_data)
x_test = vectorize_sequences(test_data)

print("x_train ", x_train.shape)
print("x_test ", x_test.shape)
```

Перекодируем номера классов в датасете в векторы вероятностей.

```
one_hot_train_labels = to_categorical(train_labels)
one_hot_test_labels = to_categorical(test_labels)

print("one_hot_train_labels ", one_hot_train_labels.shape)
print("one_hot_test_labels ", one_hot_test_labels.shape)
```

Выделим часть данных из обучающей выборки для валидации.

```
x_val = x_train[:1000]
partial_x_train = x_train[1000:]
y_val = one_hot_train_labels[:1000]
```

```
partial_y_train = one_hot_train_labels[1000:]

print("x_val ", x_val.shape)
print("y_val ", y_val.shape)
print("partial_x_train ", partial_x_train.shape)
print("partial_y_train ", partial_y_train.shape)
```

1.2. Создание базовой полносвязной модели. Подготовим референсную модель на основе двух скрытых полносвязных слоев.

```
baselinemodel = models.Sequential()
baselinemodel.add(Dense(256, kernel_regularizer=regularizers.l1(0.001),
                        activation='relu', input_shape=(Num_words,)))
baselinemodel.add(Dropout(0.5))
baselinemodel.add(Dense(256, kernel_regularizer=regularizers.l1(0.001),
                        activation='relu'))
baselinemodel.add(Dropout(0.5))
baselinemodel.add(Dense(11, activation='softmax'))
baselinemodel.summary()
```

Скомпилируем и обучим рефернсную модель на десяти эпохах.

```
NumEpochs = 10
BatchSize = 512

baselinemodel.compile(  optimizer='rmsprop',
                        loss='categorical_crossentropy',
                        metrics=['accuracy'])
history = baselinemodel.fit(partial_x_train, partial_y_train,
                            epochs=NumEpochs,
                            batch_size=BatchSize,
                            validation_data=(x_val, y_val))
```

Проверим качество работы рефернсной модели на тестовых данных.

```
results = baselinemodel.evaluate(x_val, y_val)
print("_"*100)
print("Test Loss and Accuracy")
print("results ", results)
```

Визуализируем динамику изменения метрики качества по эпохам обучения.

```
plt.clf()
acc = history.history['accuracy']
epochs = range(1, len(acc) + 1)
val_acc = history.history['val_accuracy']
plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```

1.3. Реализация рекуррентной модели. Заново подготовим данные для обучения, дополнив при необходимости последовательности нулями до максимального размера.

```
x_val = train_data[:1000]
partial_x_train = train_data[1000:]
y_val = one_hot_train_labels[:1000]
partial_y_train = one_hot_train_labels[1000:]

padded_x_val = tf.keras.utils.pad_sequences(x_val, padding="post")
padded_partial_x_train = tf.keras.utils.pad_sequences(partial_x_train,
                                                      padding="post")
maxLen = padded_x_val[0].shape[0]
```

Реализуем простую рекуррентную сеть на основе одного LSTM слоя. Векторизацию данных осуществим автоматически с помощью слоя Embedding. Дополнительно включим режим маскирования, чтобы сеть игнорировала незначащие нули, добавленные методом pad_sequences.

```
numFeatures = 40
numHidden = 45
numClasses = 11
vocabSize = Num_words

model2 = models.Sequential()
model2.add(Embedding(vocabSize,
                     numFeatures,
                     input_length=maxLen,
                     mask_zero=True)) # игнорировать нули от padding
model2.add(Bidirectional(LSTM(  numHidden,
                                return_sequences=False,
                                recurrent_dropout=0.1,
                                dropout=0.1)))
model2.add(layers.Dense(11, activation='softmax'))
model2.summary()
```

Проведем обучения рекуррентной модели на небольшом количестве эпох, чтобы по возможности избежать её быстрого переобучения.

```
NumEpochs = 4
BatchSize = 512

model2.compile( optimizer='adam',
                loss='categorical_crossentropy',
                metrics=['accuracy'])
history = model2.fit(   padded_partial_x_train, partial_y_train,
                        epochs=NumEpochs,
                        batch_size=BatchSize,
                        validation_data=(padded_x_val, y_val))
```

Сохраним модель на диске и визуализируем процесс её обучения.

```
model2.save('./recur_text_classifier.keras')
```

```
plt.clf()

acc = history.history['accuracy']
epochs = range(1, len(acc) + 1)
val_acc = history.history['val_accuracy']
plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```

§ 2. Применение LSTM в Python для задач регрессии

Итоговое значение для регрессионной модели мы будем получать с помощью одного нейрона с линейной функцией активации, а в качестве функции потерь воспользуемся среднеквадратической ошибкой.

```
numFeatures  = ____  # число признаков (число входов сети).
numHiddenUnits  = 128  # число скрытых блоков LSTM (попробуйте 64 и 256)
numResponses  = 1  # число выходов для слоя регрессии --- одно значение
numtimesteps    = None # Настройка на любую длину последовательностей

model = Sequential()  # строим модель на основе стека слоев
model.add(Masking(mask_value=0.,
                  input_shape=(numtimesteps, numFeatures)
model.add(LSTM( numHiddenUnits,
    return_sequences=True,
                recurrent_dropout = 0.2,
                dropout = 0.2 )  )
model.add(LSTM( numHiddenUnits/2,
    return_sequences=False,
                recurrent_dropout = 0.5,
                dropout = 0.5)  )
model.add(Dense(numResponses))

maxEpochs = 40          # число эпох для обучения
miniBatchSize = ____    # мини блоки обучающей выборки
model.compile(  loss='mean_squared_error',
                optimizer='adam',
                metrics =['mean_squared_error'])
history = model.fit(train_X, train_y,
     epochs=maxEpochs,
     batch_size=miniBatchSize,
     validation_split=0.2,
     verbose=2, shuffle=False)
```

§ 3. Практические задания

Задание № 1: Классификация текстов.

1. Реализовать LSTM модель для классификации 5 типов коротких текстов на английском. За основу можно взять модель из примера.
2. Добиться чтобы рекуррентная модель стабильно превосходила по качеству базовую модель. Варьировать: количество нейронов, параметры dropout, количество эпох для обучения.
3. Упорядочить обучающую выборку по длине текстов и найти оптимальное разбиение на мини блоки (приводящее к минимальному расширению нулями в пределах мини выборки). Обязательно проверить, что в модели включено маскирование нулей.
4. Сравнить время обучения модели на упорядоченных данных с минимальным числом добавленных нулей и модели с дополнением всех последовательностей до одного размера.
5. Как можно улучшить результаты? Попробуйте использовать метод ранней остановки и регуляризацию.

Задание № 2: Предсказание времени отказа для ракетных двигателей.

1. Скачать данные с сайта НАСА https://ti.arc.nasa.gov/c/6/
2. В первой колонке указаны идентефикаторы двигателей. Нужно объединить все данные с одинаковым идентификатором в отдельные последовательности.
3. Использовать данные из колонок 3 – 26 в качестве данных для входов.
4. Для каждого двигателя посчитать RUL = max значения его второй колонки (сколько тактов проработал двигатель).
5. В качестве эталонных данных для регрессии использовать прогноз по оставшимся рабочим циклам двигателя. Получить его как разность RUL и значений во второй колонке. Нейронная сеть в ответ на последовательность данных телеметрии должна выдать одно значение — сколько двигателю осталось проработать после последней записи.
6. Удалить из выборки константные данные. Нормализовать данные, и опционально обрезать эталонные значения на уровне 150 тактов двигателей. То есть оставить последние 150 записей для всех проработавших дольше двигателей.
7. Подготовить обучающую выборку, выделяя из полных данных телеметрии подпоследовательности длины от 50 до 150.
8. Упорядочить обучающую выборку по длине последовательностей и найти оптимальное разбиение на мини блоки.
9. Реализовать регрессионную модель, предусмотрев маскирование нулевых данных. За основу взять рассмотренный пример, дополнив его новыми слоями при желании, а также подбирая оптимальные настройки dropout.
10. Обучить сеть и проверить качество предсказаний. Сравнить по времени с обучением без упорядочивания по длине.
11. Как можно улучшить результаты? Попробуйте использовать метод ранней остановки и batch normalization.