

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«СИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ТЕЛЕКОММУНИКАЦИЙ И ИНФОРМАТИКИ»

ЛАБОРАТОРНАЯ РАБОТА № 4
Разработка нейронной сети

Выполнил:
Студент группы: ИП-715
Винтер А.В.

Проверил: ассистент кафедры ПМиК
Морозова К.И.

Оглавление

1. Текст задания
2. Описание основных функций
3. Результат работы программы
4. Код программы

Текст задания.

Целью данной лабораторной работы является разработка нейронной сети для решения задачи классификации или регрессии в зависимости от набора данных в рамках варианта. Лабораторная работа предполагает разработку на языке программирования Python с использованием библиотеки Keras.

Вариант задания:

- 3) Определение эмоционального окраса рецензии фильма (IMDB movie review sentiment classification dataset)

При разработке нейронной сети следует соблюсти наличие необходимых составляющих исходя из следующего варианта:

- 1) Нейросеть должна состоять из трёх полносвязных слоёв, обязательное использование Dropout, в качестве оптимизатора использовать Adam;

Выбор количества нейронов на всех внутренних слоях, функций активации и других параметров должен быть обусловлен оптимальностью работы модели.

Для защиты лабораторной работы следует обосновать выбор значений дополнительных параметров и продемонстрировать работу обученной нейронной сети. Обоснование должно включать в себя демонстрацию качества работы сети на валидационном наборе данных в процессе обучения. Параметры выбираются те, на которых валидация даёт наилучший результат.

Описание основных функций

Keras-это библиотека Python с открытым исходным кодом для легкого построения нейронных сетей.

`imdb.load_data(num_words=10000)` - Загружает набор данных IMDB. Это набор данных из 50 000 обзоров фильмов из IMDB, помеченных (положительный / отрицательный). Рецензии были предварительно обработаны, и каждая рецензия является кодируется в виде списка индексов слов (целых чисел).

```
data = np.concatenate((training_data, testing_data), axis = 0)
targets = np.concatenate((training_targets, testing_targets), axis = 0)
```

Для избежания разделения test train 50/50, мы сразу же объединим данные в data и targets после загрузки, чтобы позже выполнить разделение.

`def vectorize(sequences, dimension = 10000):` - векторизуем каждый отзыв и заполняем его нулями, чтобы он содержал ровно 10 000 чисел. Заполняем каждый обзор, который короче 10 000, нулями.

```
data = vectorize(data)
targets = np.array(targets).astype("float32")
test_x = data[:10000]
test_y = targets[:10000]
train_x = data[10000:]
train_y = targets[10000:]
```

Разделили данные на train и test наборы. Train набор будет содержать 40.000 обзоров, а test набор - 10.000

`model = models.Sequential()` - Создание последовательной модели

```

model.add(layers.Dropout(0.2, input_shape = (10000,)))
model.add(layers.Dense(50, activation = "relu", input_shape = (10000,)))
model.add(layers.Dropout(0.3, input_shape = (50,)))
model.add(layers.Dense(50, activation = "relu"))
model.add(layers.Dropout(0.2, input_shape = (50,)))
model.add(layers.Dense(1, activation = "sigmoid"))
model.summary()
model.compile(optimizer = "adam", loss = "binary_crossentropy", metrics = ["accuracy"])

```

Далее добавили входные, скрытые и выходные слои. Между ними мы используем отсеивание, чтобы предотвратить переобучение. Коэффициент отсеивания равен от 0.2 до 0.5

```

model.compile(optimizer = "adam", loss = "binary_crossentropy", metrics = ["accuracy"])

```

Используем оптимизатор "adam", алгоритм, который изменяет веса и смещения во время тренировки. Мы также Выбираем двоичную кроссэнтропию в качестве потери (потому что мы имеем дело с двоичной классификацией) и точность в качестве нашей оценочной метрики.

```

results = model.fit(train_x, train_y, epochs = 10, batch_size = 500, validation_split = 0.2) - Обучение сети

```

Тренируем нашу модель с размером пакета 500, для 10 «эпох»

Размер пакета определяет количество выборок, которые будут распространяться по сети, а эпоха - это итерация по всем обучающим данным

```

score = model.evaluate(test_x, test_y) - возвращает значение потерь и значения метрик для модели в тестовом режиме

```

#Запуск распознавателя:

```

prediction = model.predict(test_x) - генерирует выходные прогнозы для входных выборок.

```

Результат работы программы

Model: "sequential_1"

Layer (type)	Output Shape	Param #
dropout_3 (Dropout)	(None, 10000)	0
dense_3 (Dense)	(None, 50)	500050
dropout_4 (Dropout)	(None, 50)	0
dense_4 (Dense)	(None, 50)	2550
dropout_5 (Dropout)	(None, 50)	0
dense_5 (Dense)	(None, 1)	51
Total params: 502,651		
Trainable params: 502,651		
Non-trainable params: 0		

Epoch 1/10

64/64 [=====] - 5s 73ms/step - loss: 0.4418 - accuracy: 0.7996 - val_loss: 0.2660 - val_accuracy: 0.8950

Epoch 2/10

64/64 [=====] - 5s 74ms/step - loss: 0.2684 - accuracy: 0.8913 - val_loss: 0.2442 - val_accuracy: 0.9044

Epoch 3/10

64/64 [=====] - 5s 75ms/step - loss: 0.2231 - accuracy: 0.9116 - val_loss: 0.2486 - val_accuracy: 0.9014

Epoch 4/10

64/64 [=====] - 5s 75ms/step - loss: 0.1924 - accuracy: 0.9252 - val_loss: 0.2594 - val_accuracy: 0.8944

Epoch 5/10

64/64 [=====] - 5s 72ms/step - loss: 0.1655 - accuracy: 0.9348 - val_loss: 0.2648 - val_accuracy: 0.8959

Epoch 6/10

64/64 [=====] - 5s 74ms/step - loss: 0.1423 - accuracy: 0.9450 - val_loss: 0.2685 - val_accuracy: 0.8971

Epoch 7/10

64/64 [=====] - 5s 73ms/step - loss: 0.1228 - accuracy: 0.9533 - val_loss: 0.2827 - val_accuracy: 0.8944

Epoch 8/10

64/64 [=====] - 5s 74ms/step - loss: 0.1084 - accuracy: 0.9592 - val_loss: 0.2901 - val_accuracy: 0.8915

Epoch 9/10

64/64 [=====] - 5s 72ms/step - loss: 0.1003 - accuracy: 0.9622 - val_loss: 0.3058 - val_accuracy: 0.8889

Epoch 10/10

64/64 [=====] - 4s 70ms/step - loss: 0.0895 - accuracy: 0.9678 - val_loss: 0.3239 - val_accuracy: 0.8884

313/313 [=====] - 1s 3ms/step - loss: 0.3608 - accuracy: 0.8825

Test потери: 36.08%

Test точность: 88.25%

Ожидаемое	Фактическое
-----------	-------------

0.9999791	1.000000
-----------	----------

0.0000008	0.000000
-----------	----------

0.0000328	0.000000
-----------	----------

0.9999896	1.000000
-----------	----------

0.0000523	0.000000
-----------	----------

0.0868732	0.000000
-----------	----------

0.9993737	1.000000
-----------	----------

0.0000000	0.000000
-----------	----------

0.9999524	1.000000
-----------	----------

0.0549214	0.000000
-----------	----------

Код программы

```
import numpy as np

from keras import models
from keras import layers
from keras.datasets import imdb

(training_data, training_targets), (testing_data, testing_targets) = imdb.load_data(
    num_words=10000)
data = np.concatenate((training_data, testing_data), axis = 0)
targets = np.concatenate((training_targets, testing_targets), axis = 0)

def vectorize(sequences, dimension = 10000):
    results = np.zeros((len(sequences), dimension))
    for i, sequence in enumerate(sequences):
        results[i, sequence] = 1
    return results

def toFixed(numObj, digits =0):
    return f"{numObj:.{digits}f}"

data = vectorize(data)
targets = np.array(targets).astype("float32")
test_x = data[:10000]
test_y = targets[:10000]
train_x = data[10000:]
train_y = targets[10000:]

#Создание последовательной модели, добавление уровней сети и компиляция модели:
model = models.Sequential()

model.add(layers.Dropout(0.2, input_shape = (10000,)))
model.add(layers.Dense(50, activation = "relu", input_shape = (10000,)))

model.add(layers.Dropout(0.3, input_shape = (50,)))
model.add(layers.Dense(50, activation = "relu"))

model.add(layers.Dropout(0.2, input_shape = (50,)))
model.add(layers.Dense(1, activation = "sigmoid"))
model.summary()
model.compile(optimizer = "adam", loss = "binary_crossentropy", metrics = ["accuracy"])

#Обучение сети:
results = model.fit(train_x, train_y, epochs = 10, batch_size = 500, validation_split = 0.2)

#Возвращает значение потерь и значения метрик для модели в тестовом режиме.
score = model.evaluate(test_x, test_y)
print("Test потери: %.2f%%" % (score[0] * 100))
print("Test точность: %.2f%%" % (score[1] * 100))
```

```
#Запуск распознавателя
prediction = model.predict(test_x)
print("Ожидаемое | Фактическое")
for i in range(10):
    print('%.7f | %f' % (prediction[i], test_y[i]))
```