**Модель полносвязной нейроннойсети для задачи бинарной классификации**

В данной работе рассматривается модель, которая будет классифицировать обзор фильма как *позитивный* или *негативный* на основе текста обзора. Это пример *бинарной* классификации (по двум классам), важной, и широко применяющейся задачи машинного обучения.

Воспользуемся набором данных IMDB, который содержит тексты 50000 обзоров фильмов из [Internet Movie Database](https://www.imdb.com/). Они разделены на 25000 обзоров для обучения, и 25000 для проверки модели. Обучающие и проверочные наборы данных *сбалансированы*, т.е. содержат одинаковое количество позитивных и негативных обзоров.

Для создания и обучения модели используется высокоуровневый API [tf.keras](https://www.tensorflow.org/guide/keras), содержащийся в библиотеке моделей в TensorFlow. Более сложная по структуре классификация текста при помощи [tf.keras](https://www.tensorflow.org/api_docs/python/tf/keras) рассматривается в [Руководстве по классификации текстов](https://developers.google.com/machine-learning/guides/text-classification/).

Данное руководство рассчитано, что приведенные фрагменты запускаются последовательно в Jupiter Notebook. При запуске в других IDE (где запускается программа полностью: Spyder, PyCharm) некоторые фрагменты можно опустить. Предполагается, что установлена библиотека Tensorflow версии 2.0 или выше (она включает библиотеку keras в качестве составной части). Установка библиотек описана в приложении к лабораторной работе 3.

**1. Подключение необходимых библиотек**

Загрузим необходимые библиотеки и проверим версию TensorFlow (в вашем случае версия может быть другой).

import tensorflow as tf  
from tensorflow import keras  
  
import numpy as np  
  
print(tf.\_\_version\_\_)

2.0.0 # зависит от установки

**2. Загрузка набора данных IMDB**

Набор данных IMDB поставляется в составе TensorFlow и может быть загружен при помощи метода load\_data. Он уже подготовлен таким образом, что обзоры (последовательности слов) конвертированы в последовательность целых чисел, где каждое число представляет конкретное слово в массиве. Кодировка слов выполняется в соответствии со словарем на 10000 наиболее употребимых слов.

Для загрузки набора данных используется следующий код:

imdb = keras.datasets.imdb  
  
(train\_data, train\_labels), (test\_data, test\_labels) = imdb.load\_data(num\_words=10000)

Аргумент num\_words=10000 оставляет только 10000 наиболее часто встречающих слов из обучающего набора. Все редкие слова из набора исключаются. Это поможет ограничить объем данных разумными пределами.

**3. Изучение данных**

Рассмотрим загруженные данные.

Каждая метка *label* является целым числом 0 или 1:

*0 - негативный обзор, 1 - позитивный.*

print("Обучающих записей: {}, меток: {}".format(len(train\_data), len(train\_labels)))

Обучающих записей: 25000, меток: 25000

Данные каждого обзора - это массив целых чисел, которые представляют слова из словаря. Вот пример того, как представлен первый обзор:

print(train\_data[0])

[1, 14, 22, 16, 43, 530, 973, 1622, 1385, 65, 458, 4468, 66, 3941, 4, 173, 36, 256, 5, 25, 100, 43, 838, 112, 50, 670, 2, 9, 35, 480, 284, 5, 150, 4, 172, 112, 167, 2, 336, 385, 39, 4, 172, 4536, 1111, 17, 546, 38, 13, 447, 4, 192, 50, 16, 6, 147, 2025, 19, 14, 22, 4, 1920, 4613, 469, 4, 22, 71, 87, 12, 16, 43, 530, 38, 76, 15, 13, 1247, 4, 22, 17, 515, 17, 12, 16, 626, 18, 2, 5, 62, 386, 12, 8, 316, 8, 106, 5, 4, 2223, 5244, 16, 480, 66, 3785, 33, 4, 130, 12, 16, 38, 619, 5, 25, 124, 51, 36, 135, 48, 25, 1415, 33, 6, 22, 12, 215, 28, 77, 52, 5, 14, 407, 16, 82, 2, 8, 4, 107, 117, 5952, 15, 256, 4, 2, 7, 3766, 5, 723, 36, 71, 43, 530, 476, 26, 400, 317, 46, 7, 4, 2, 1029, 13, 104, 88, 4, 381, 15, 297, 98, 32, 2071, 56, 26, 141, 6, 194, 7486, 18, 4, 226, 22, 21, 134, 476, 26, 480, 5, 144, 30, 5535, 18, 51, 36, 28, 224, 92, 25, 104, 4, 226, 65, 16, 38, 1334, 88, 12, 16, 283, 5, 16, 4472, 113, 103, 32, 15, 16, 5345, 19, 178, 32]

Разные обзоры фильмов содержат разное количество слов. Следующий код ниже показывает количество слов в первом и втором образце (обзоре ) набора данных.

print(len(train\_data[0]), len(train\_data[1]))

(218, 189)

Поскольку нейросеть может принимать только данные одинаковой длины, то необходимо скорректировать данные (методы корректировке описаны в следующем подразделе).

Можно для наглядности конвертировать целые числа представления обзора обратно в текст. Для этого используем следующую вспомогательную функцию decode\_review, которая обращается к словарю для обратного представления индекса слова в само слово:

# Назначим словарь, который будет отображать слова из массива данных  
word\_index = imdb.get\_word\_index()  
  
# Зарезервируем первые несколько значений  
word\_index = {k:(v+3) for k,v in word\_index.items()}  
word\_index["<PAD>"] = 0  
word\_index["<START>"] = 1  
word\_index["<UNK>"] = 2  # Вместо редких слов, не вошедших в набор из 10000, будет указано UNK  
word\_index["<UNUSED>"] = 3  
  
reverse\_word\_index = dict([(value, key) for (key, value) in word\_index.items()])  
  
def decode\_review(text):  
    return ' '.join([reverse\_word\_index.get(i, '?') for i in text])

Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/imdb\_word\_index.json

1646592/1641221 [==============================] - 0s 0us/step

Теперь можно использовать функцию decode\_review для отображения оригинального текста первого обзора фильма:

decode\_review(train\_data[0])

"<START> this film was just brilliant casting location scenery story direction everyone's really suited the part they played and you could just imagine being there robert <UNK> is an amazing actor and now the same being director <UNK> father came from the same scottish island as myself so i loved the fact there was a real connection with this film the witty remarks throughout the film were great it was just brilliant so much that i bought the film as soon as it was released for <UNK> and would recommend it to everyone to watch and the fly fishing was amazing really cried at the end it was so sad and you know what they say if you cry at a film it must have been good and this definitely was also <UNK> to the two little boy's that played the <UNK> of norman and paul they were just brilliant children are often left out of the <UNK> list i think because the stars that play them all grown up are such a big profile for the whole film but these children are amazing and should be praised for what they have done don't you think the whole story was so lovely because it was true and was someone's life after all that was shared with us all"

**4. Подготовка данных**

Обзоры фильмов из массива целых чисел должны быть конвертированы в тензоры одинакового размера перед загрузкой в модель нейронной сети. Эта конвертация может быть сделана двумя способами:

* Конвертировать массивы в векторы 0 и 1 в формате *one-hot encoding*. Например, последовательность [3, 5] станет 10000-мерным вектором (определяется количеством возможных слов в словаре), полностью состоящим из нулей кроме позиций 3 и 5, которые будут заполнены единицами. Такой подход очень требователен к объему памяти, так как размер входной матрицы будет num\_words \* num\_reviews
* Сделать все массивы одинаковыми по длине, а затем создать тензор целых чисел с указанием max\_length \* num\_reviews. Для такого подхода можно использовать *Embedding* ("Встроенный") слой, который может использовать эти параметры в качестве первого слоя нашей сети

В данной работе используется второй способ (первый способ описан в лекции). Для приведения всех обзоров фильмов к одинаковой длине используется функция [pad\_sequences](https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/sequence/pad_sequences), чтобы привести все длины к одному значению (256 элементов) дополнением последовательности до нужного размера нулями:

train\_data = keras.preprocessing.sequence.**pad\_sequences**(train\_data,  
value=word\_index["<PAD>"],  
padding='post',  
maxlen=256)  
  
test\_data = keras.preprocessing.sequence.**pad\_sequences**(test\_data, value=word\_index["<PAD>"],  
padding='post',  
maxlen=256)

Здесь

value - значение, которым дополняется каждый обзор (вектор train\_data[i]) до одинаковой длины. В примере word\_index["<PAD>"] = 0

padding - место дополнения (pre - в начале, post - в конце последовательности)

maxlen - длина последовательности

Убедимся, что обзоры с индексами 0 и 1 после преобразования будут иметь одинаковую длину 256 элементов:

print(len(train\_data[0]), len(train\_data[1]))

(256, 256)

Массив слов каждого обзора после преобразования будет дополнен нулями до фиксированного размера 256. В этом можно убедиться при выводе первого обзора:

print(train\_data[0])

[ 1 14 22 16 43 530 973 1622 1385 65 458 4468 66 3941

4 173 36 256 5 25 100 43 838 112 50 670 2 9

35 480 284 5 150 4 172 112 167 2 336 385 39 4

172 4536 1111 17 546 38 13 447 4 192 50 16 6 147

2025 19 14 22 4 1920 4613 469 4 22 71 87 12 16

43 530 38 76 15 13 1247 4 22 17 515 17 12 16

626 18 2 5 62 386 12 8 316 8 106 5 4 2223

5244 16 480 66 3785 33 4 130 12 16 38 619 5 25

124 51 36 135 48 25 1415 33 6 22 12 215 28 77

52 5 14 407 16 82 2 8 4 107 117 5952 15 256

4 2 7 3766 5 723 36 71 43 530 476 26 400 317

46 7 4 2 1029 13 104 88 4 381 15 297 98 32

2071 56 26 141 6 194 7486 18 4 226 22 21 134 476

26 480 5 144 30 5535 18 51 36 28 224 92 25 104

4 226 65 16 38 1334 88 12 16 283 5 16 4472 113

103 32 15 16 5345 19 178 32 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0]

**5. Создание модели**

Модель нейтронной сети включает стек слоев. При создании модели необходимо определить:

* Сколько слоев будет использовано в модели
* Сколько *скрытых нейронов* будет использовано в каждом слое

В рассматриваемом примере, входные данные состоят из массива слов (целых чисел). Получаемые предсказания представлены в виде меток 0 или 1. Сконструируем следующую модель и покажем ее структуру:

# Размер входных данных - количество слов, использованных в обзорах фильмов (10000 слов)  
vocab\_size = 10000  
  
model = keras.Sequential()  
model.add(keras.layers.Embedding(vocab\_size, 16, input\_shape=(None,)))  
model.add(keras.layers.GlobalAveragePooling1D())  
model.add(keras.layers.Dense(16, activation=tf.nn.relu))  
model.add(keras.layers.Dense(1, activation=tf.nn.sigmoid))  
  
model.summary()

Model: "sequential"

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param #

=================================================================

embedding (Embedding) (None, None, 16) 160000

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

global\_average\_pooling1d (Gl (None, 16) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense (Dense) (None, 16) 272

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense\_1 (Dense) (None, 1) 17

=================================================================

Total params: 160,289

Trainable params: 160,289

Non-trainable params: 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

В модель включены следующие слои:

1. Первый слой Embedding принимает преобразованный массив обзоров и ищет соответствующий вектор для каждой пары слово/число. Модель обучается на этих векторах. Векторы увеличивают размерность получаемого массива на 1.

При добавлении слоя необходимо указать 3 аргумента:

* **input\_dim**: это размер словаря текстовых данных (в нашем случае это vocab\_size = 10000).
* **output\_dim**: это размер векторного пространства, в которое будут преобразованы слова входных векторов. Он определяет размер выходного вектора этого слоя для каждого слова (в нашем случае выбрано 16). Можно проверить различные значения для конкретной задачи.
* **input\_length**: Это длина входных последовательностей (число элементов, число образцов). В нашем случае число элементов 256, но как правило используется None и модель будет определять размер последовательности автоматически. Как правило, не указывают и число образцов, которое также определяется автоматически. Это делает модель универсальной под любой размер входных данных.

1. Следующий слой GlobalAveragePooling1D возвращает получаемый вектор заданной длины для каждого примера, усредняя размер ряда. Это позволит модели легко принимать данные разной длины
2. Следующий полносвязный слой Dense с 16 скрытыми нейронами принимает на вход вектор предыдущего слоя
3. Последний слой также является полносвязным, но всего с одним выходящим узлом. При помощи функции активации sigmoid (Сигмоида) на выходе будем получать число с плавающей запятой между 0 и 1, которое будет показывать вероятность отнесения отзыва к одному из классов

**Скрытые слои**

Вышеописанная модель имеет 2 промежуточных или *скрытых* слоя, между входом и выходом данных. Количество выходов (узлов или нейронов) является размером репрезентативного пространства слоя. Другими словами, количество степеней свободы, которая разрешена сети во время обучения.

Если модель имеет больше скрытых узлов, и/или больше слоев, то тогда нейросеть может обучиться более сложным представлениям. Однако в этом случае это будет дороже с точки зрения вычислительных ресурсов и может привести к переобучению, т.е. обучению распознавания нежелательных паттернов - паттернов, которые улучшают показатели на тренировочных данных, но не на проверочных.

**6. Определение функции потерь и оптимизатора**

Для модели необходимо указать функцию потерь и оптимизатор для обучения. Поскольку задача является примером бинарной классификации и модель будет показывать вероятность (выходной слой с одним выходом с сигмоидой в качестве функции активации), то мы воспользуемся функцией потерь binary\_crossentropy (Перекрестная энтропия). Она измеряет "дистанцию" между распределениями вероятностей, или, как в нашем случае, между эталоном и предсказаниями

Это не единственный выбор для функции потерь: можно, например, выбрать mean\_squared\_error (среднеквадратическая ошибка). Но обычно binary\_crossentropy лучше справляется с вероятностями. Функция потерь среднеквадратичской ошибки часто используется в задачах регрессии, как будет показано в последней работе.

В качестве оптимизатора будем использовать оптимизатор Адама. Выполним настройку (компиляцию) модели:

model.compile(optimizer='adam',  
              loss='binary\_crossentropy',  
              metrics=['accuracy'])

**7. Создание проверочного набора данных**

Во время обучения следует проверить точность модели на данных, которых она еще не видела. Для этого создадим *проверочный набор* данных, выделив 10000 примеров из оригинального тренировочного сета в отдельный набор.

Настройка модели производится с использованием только данных для обучения, и только потом используется проверочный набор для оценки точности.

x\_val = train\_data[:10000]  
partial\_x\_train = train\_data[10000:]  
  
y\_val = train\_labels[:10000]  
partial\_y\_train = train\_labels[10000:]

**8. Обучение модели**

Начнем обучение модели с 40 эпох при помощи мини-батчей (пакетов) по 512 образцов. Это означает, что выполняется 40 итераций (или проходов) по всем образцам данных в тензорах x\_train и y\_train:

history = model.fit(partial\_x\_train,  
                    partial\_y\_train,  
                    epochs=40,  
                    batch\_size=512,  
                    validation\_data=(x\_val, y\_val),  
                    verbose=1)

Train on 15000 samples, validate on 10000 samples

Epoch 1/40

15000/15000 [==============================] - 1s 61us/sample - loss: 0.6922 - acc: 0.5586 - val\_loss: 0.6909 - val\_acc: 0.5502

...

Epoch 39/40

15000/15000 [==============================] - 1s 44us/sample - loss: 0.1033 - acc: 0.9708 - val\_loss: 0.3038 - val\_acc: 0.8838

Epoch 40/40

15000/15000 [==============================] - 1s 44us/sample - loss: 0.0992 - acc: 0.9732 - val\_loss: 0.3062 - val\_acc: 0.8840

Построим временной график точности и потерь

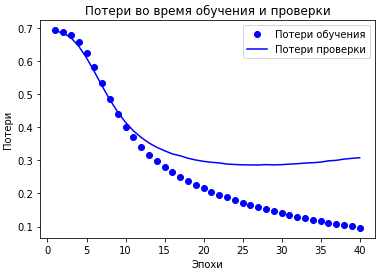
Метод model.fit() возвращает объект History, который содержит все показатели, которые были записаны в лог во время обучения:

history\_dict = history.history  
history\_dict.keys()

dict\_keys(['val\_accuracy', 'accuracy', 'val\_loss', 'loss'])

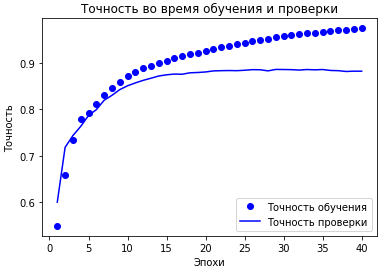
Здесь всего четыре показателя, по одному для каждой отслеживаемой метрики во время обучения и проверки. Мы можем использовать их, чтобы построить графики потерь и точности обоих стадий для сравнения:

import matplotlib.pyplot as plt  
  
acc = history.history['accuracy']  
val\_acc = history.history['val\_accuracy']  
loss = history.history['loss']  
val\_loss = history.history['val\_loss']  
  
epochs = range(1, len(acc) + 1)  
  
# "bo" означает "blue dot", синяя точка  
plt.plot(epochs, loss, 'bo', label='Потери обучения')  
# "b" означает "solid blue line", непрерывная синяя линия  
plt.plot(epochs, val\_loss, 'b', label='Потери проверки')  
plt.title('Потери во время обучения и проверки')  
plt.xlabel('Эпохи')  
plt.ylabel('Потери')  
plt.legend()  
  
plt.show()



plt.clf()   # Очистим график

plt.plot(epochs, acc, 'bo', label='Точность обучения')  
plt.plot(epochs, val\_acc, 'b', label='Точность проверки')  
plt.title('Точность во время обучения и проверки')  
plt.xlabel('Эпохи')  
plt.ylabel('Точность')  
plt.legend()  
  
plt.show()



На графиках точками отмечены потери и точность модели во время обучения, а линией - во время проверки.

Обратите внимание на то, что потери во время обучения *уменьшаются*, а точность – *увеличивается* с каждой следующей эпохой. Это вполне ожидаемо, поскольку использовался градиентный спуск *gradient descent* - он минимизирует показатели потерь с каждой итерацией настолько быстро, насколько это возможно.

Но если посмотреть на потери и точность во время проверки модели: после приблизительно 20 эпох они находятся на пике, а далее уменьшаются. Это явный пример переобучения: модель показывает более лучшие показатели на данных для обучения, нежели на новых, которых она еще не видела. После этого момента модель начинает переоптимизироваться и обучается представлениям, которые *свойственны* только данным обучения. Таким образом, модель не учится *обобщать* новые, проверочные данные.

Именно здесь можно предотвратить переобучение просто прекратив тренировку сразу после 20 эпох обучения. Это можно сделать автоматически при помощи *callback*, функции обратного вызова.

**9. Оценка точности модели**

Теперь когда обучение прошло успешно, проверим точность модели на 10000 образцах из контрольного набора данных. Метод проверки будет возвращать 2 значения: потери *loss* (число, которое показывает ошибку, чем оно ниже, тем лучше), и точность *accuracy (*чем она выше, тем лучше).

results = model.evaluate(test\_data, test\_labels)  
print(results)

25000/1 [=====================================================================================

. . .

25000/25000 [====================]] - 1s 34us/sample - loss: 0.3190 - accuracy: 0.8722

[0.32863121019363406, 0.8722]

Как видно, этот достаточно наивный подход к построению модели достиг точности около 87%. Если бы мы использовали более сложные методы, то модель приблизилась бы к отметке в 95%.

**10. Сохранение и восстановление модели**

Сохраним модель целиком в единый файл, который будет содержать все веса обученной модели, конфигурацию модели и оптимизатор конфигурации. Это позволит впоследствии восстановить модель и использовать ее для решения задачи классификации без предварительного обучения.

В Keras есть встроенный формат для сохранения модель при помощи стандарта [HDF5](https://en.wikipedia.org/wiki/Hierarchical_Data_Format). Сохраним модель.

model.save('путь\model\_one\_class.h5')

Создадим новую модель из сохраненного файла:

new\_model = keras.models.load\_model('путь\ model\_one\_class.h5')

Проверим созданную модель из файла

loss, acc = new\_model.evaluate(test\_data, test\_labels)

print("Восстановленная модель, точность: {:5.2f}%".format(100\*acc))

25000/1 [===============================================================================

. . .

===========================] - 1s 40us/sample - loss: 0.3496 - accuracy: 0.8722

Восстановленная модель, точность: 87.22%

Видно, что восстановленная модель дает такие же результаты, как и ранее обученная модель.

**Задание.** Постройте модель заданной архитектуры, обучите и проверьте ее точность. Сохраните модель. Восстановите модель и сравните точность обеих моделей.

**Дополнительное задание**: Измените параметры архитектуры модели сети и выберите оптимальные значения

* В данном примере использовался один скрытый слой Dense. Попробуйте использовать два или три и посмотрите, как это повлияет на точность на этапах обучения и проверки.
* Попробуйте использовать слои с большим или с меньшим количеством скрытых нейронов: 32 нейрона, 64 нейрона и т. д.
* Попробуйте вместо binary\_crossentropy использовать функцию потерь mean\_squared\_error
* Попробуйте вместо relu в промежуточном слое использовать функцию активации tanh (она была популярна на заре становления нейронных сетей).