1)

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

Keras-это API глубокого обучения, написанный на Python и работающий поверх платформы машинного обучения TensorFlow. Он был разработан с акцентом на обеспечение быстрого экспериментирования. *Способность как можно быстрее перейти от идеи к результату является ключом к проведению хороших исследований.*

Keras - это высокоуровневый API TensorFlow 2.0: доступный, высокопроизводительный интерфейс для решения задач машинного обучения, с акцентом на современное глубокое обучение. Он предоставляет существенные абстракции и строительные блоки для разработки и доставка решений машинного обучения с высокой скоростью итерации.

2)

```
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).

num_words: целое число или нет. Слова есть ранжируются по тому, как часто они встречаются (в тренировочном наборе) и только `num_words`самые частые слова сохраняются. Любое менее частое слово появится как `oov_char`значение в данных последовательности. Если Нет, все слова сохранены. По умолчанию нет, поэтому все слова сохраняются.

oov_char: int. - характер вне словаря. Слова, которые были вырезаны из-за `num_words` `или` `skip_top` ограничения будут заменены этим символом.

3)

Поскольку мы хотим избежать разделения теста train 50/50, мы сразу же объединим данные в данные и цели после загрузки, чтобы позже выполнить разделение.

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

Соедините последовательность массивов вдоль существующей оси.

Axis int, опционально
Ось, вдоль которой будут соединены массивы. Если ось отсутствует, массивы перед использованием сглаживаются. Значение по умолчанию-0.

4) Мы векторизуем каждый отзыв и заполняем его нулями, чтобы он содержал ровно 10 000 чисел. Это означает, что мы заполняем каждый обзор, который короче 10 000, нулями. Мы должны сделать это, потому что самый большой обзор почти такой же длины, и каждый вход для нашей нейронной сети должен иметь одинаковый размер. Мы также преобразуем цели в поплавки.

```
def vectorize(sequences, dimension = 10000):
```

sequences - последовательности

dimension – изменение

Определите векторизованную функцию, которая принимает вложенную последовательность объектов или массивов numpy в качестве входных данных и возвращает один массив numpy или кортеж массивов numpy. Векторизованная функция вычисляет pyfunc по последовательным кортежам входных массивов, как функция python map, за исключением того, что она использует правила вещания numpy.

results[i, sequence] = 1

набор конкретных показателей результатов[i] для первых

5)

Теперь мы разделили наши данные на тренировочный и тестовый наборы. Учебный набор будет содержать 40 000 обзоров, а тестовый набор-10 000.

```
test_x = data[:10000]
test_y = targets[:10000]
train_x = data[10000:]
train_y = targets[10000:]
```

6)

model.add(layers.Dropout(0.2, input_shape = (10000,)))

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

Начнем с определения типа модели, которую мы хотим построить. В Keras доступны два типа моделей: последовательная модель и класс моделей, используемый с функциональным API.

Далее мы просто добавим входные, скрытые и выходные слои. Между ними мы используем отсев, чтобы предотвратить переобучение

Пожалуйста, обратите внимание, что вы всегда должны использовать коэффициент отсева от 20% до 50%.

Мы используем "Dense" на каждом слое, что означает, что блоки полностью соединены. В скрытых слоях мы используем функцию relu, потому что это всегда хорошее начало и дает удовлетворительный результат большую часть времени. Не стесняйтесь экспериментировать с другими функциями активации.

На выходном слое мы используем сигмоидную функцию, которая отображает значения между 0 и 1. Обратите внимание, что мы устанавливаем входную форму на 10 000 на входном слое, потому что наши обзоры имеют длину 10 000 целых чисел. Входной слой принимает 10 000 в качестве входных данных и выводит их с формой 50.

`relu` функция

Применяет функцию активации выпрямленного линейного блока.

При значениях по умолчанию это возвращает стандартную активацию ReLU: `max(x, 0)`, элементный максимум 0 и входной тензор.

`sigmoid` функция

Сигмовидная активационная функция, `sigmoid(x) = 1 / (1 + exp(-x))`.

Применяет функцию активации сигмовидной мышцы. Для малых значений (), `sigmoid` возвращает значение, близкое к нулю, а для больших значений (>5) результат работы функции приближается к 1.

`softmax` функция

Softmax преобразует реальный вектор в вектор категориальных вероятностей.

Элементы выходного вектора находятся в диапазоне (0, 1) и суммируются до 1.

Каждый вектор обрабатывается независимо. `axis`Аргумент задает, какая Ось из входных данных функция применяется вдоль.

Softmax часто используется в качестве активации для последнего уровень классификационной сети, поскольку результат может быть интерпретирован как вероятностное распределение.

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

Теперь мы компилируем нашу модель, которая является не чем иным, как настройкой модели для обучения. Мы используем оптимизатор “Адам", 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.

7)

```
#Обучение сети:

results = model.fit(train_x, train_y, epochs = 10, batch_size = 500, validation_spl
it = 0.2)
```

Теперь мы можем тренировать нашу модель. Мы сделаем это с размером пакета 500 и только для двух эпох, потому что я признал, что модель подходит, если мы тренируем ее дольше.

Размер пакета определяет количество выборок, которые будут распространяться по сети, а эпоха - это итерация по всем обучающим данным. В общем, больший размер партии приводит к более быстрому обучению, но не всегда сходится так быстро. Меньший размер пакета медленнее в обучении, но он может сходиться быстрее. Это определенно зависит от проблемы, и вам нужно будет попробовать несколько различных значений. Если вы начинаете с проблемы в первый раз, я рекомендую сначала использовать пакет размером 32, который является стандартным размером.

8)

```
prediction = model.predict(test_x) #Генерирует выходные прогно
зы для входных выборок.
```