Functional API: руководство

Функциональный API Keras — это способ создания сложных моделей, таких, например, которые имеют несколько выходов, модели с общими слоями, или ациклические графы.

В этом руководстве предполагается, что вы уже знакомы с моделью Sequential.

Давайте рассмотрим несколько простых примеров.

Первый пример: полносвязная сеть

Модель Sequential, вероятно, является лучшей реализацией подобной сети, но нам просто нужно начать с чего-то действительно простого.

- Экземпляр слоя применяется к тензору и возвращает тензор.
- Входной и выходной тензор(ы) затем могут также использоваться для определения модели
- Такая модель обучается также как и модель Sequential

```
from keras.layers import Input, Dense

from keras.models import Model

# Этот код вернет тензор

inputs = Input(shape=(784,))

# Экземпляр слоя применяется к тензору и возвращает новый тензор

output_1 = Dense(64, activation='relu')(inputs)

output_2 = Dense(64, activation='relu')(output_1)
```

```
predictions = Dense(10, activation='softmax')(output_2)

# Мы создали модель, включающую

# один входной слой и три Dense слоя

model = Model(inputs=inputs, outputs=predictions)

model.compile(optimizer='rmsprop',

        loss='categorical_crossentropy',

        metrics=['accuracy'])

model.fit(data, labels)  # Старт обучения
```

Все модели могут вызываться как слои

С помощью функционального API вы можете легко использовать обученные модели: вы можете применять любую модель к тензору, как если бы она была слоем. Обратите внимание, что вызывая модель, вы не просто повторно используете ее архитектуру, но также повторно используете и ее веса.

```
x = Input(shape=(784,))

# Это сработает и вернет 10мерный-softmax вектор, который мы определили выше.

y = model(x)
```

Это может помочь, например, быстро создавать модели, способные обрабатывать последовательности входных данных. Вы можете превратить модель классификации изображений в модель классификации видео, изменив всего одну строку.

```
from keras.layers import TimeDistributed

# Входной тензор для последовательности из 20 кадров

# каждый из которых содержит 784-мерный вектор
```

input_sequences = Input(shape=(20, 784))

 # Применяем нашу предыдущую модель к каждому кадру во входной последовательности.

 # Выход предыдущей модели был 10-мерный softmax вектор

 # Поэтому на выходе новой модели будет последовательность из 20 10-мерных векторов

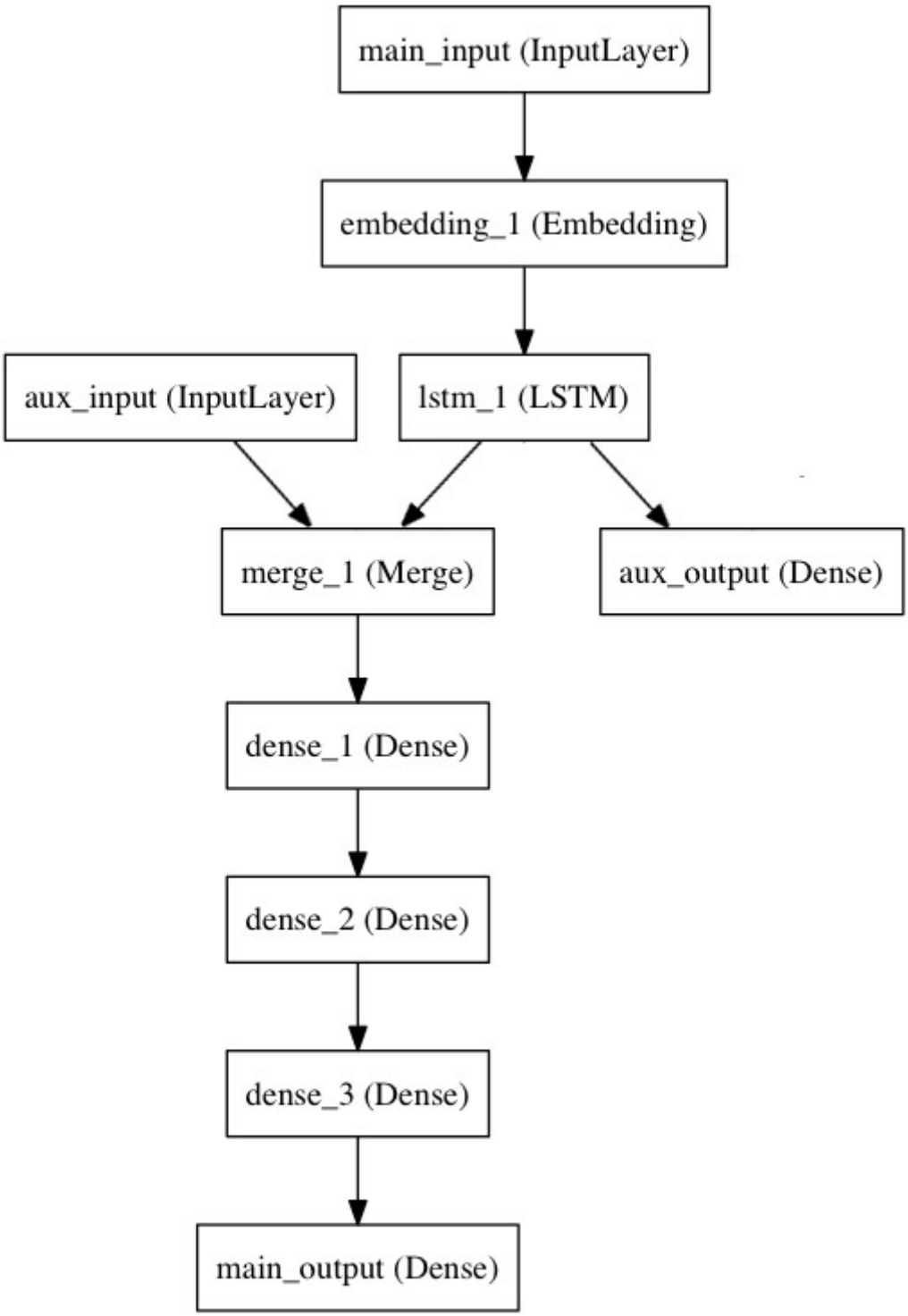
processed_sequences = TimeDistributed(model)(input_sequences)

Модели с несколькими входами и несколькими выходами

Рассмотрим хороший пример использования функционального API: модель с несколькими входами и несколькими выходами. Функциональное API позволяет легко управлять большим количеством пересекающихся потоков данных.

Давайте рассмотрим следующую модель. Мы попробуем предсказать сколько ретвитов и лайков будут получать заголовки новостей в Twitter. Основным входом в модель будет сам заголовок в виде последовательности слов. Помимо этого, наша модель будет иметь вспомогательный вход, на который будут подаваться дополнительные данные, такие как время публикации заголовка и т.п. Модель будет также иметь две функции ошибки. Использование основной функции ошибки ранее в модели является хорошим механизмом регуляризации для глубоких моделей.

Вот как выглядит наша модель:



Давайте реализуем это с помощью функционального API.

Основной вход получит заголовок в виде последовательности целых чисел (каждое целое число кодирует слово). Целые числа будет в интервале от 1 до 10000 (словарь из 10 тыс. слов), а последовательности будут длиной 100 слов.

from keras.layers **import** Input, Embedding, LSTM, Dense

from keras.models **import** Model

import numpy **as** np

np.random.seed(0) **# Зададим случайное число для генератора**

 # Вход заголовка: предназначен для получения последовательности

 # из 100 чисел в диапазоне от 1 до 10000

 # Обратите внимание, что мы можем дать имя любому слою,

#указав его в параметре name

main_input = Input(shape=(100,), dtype='int32', name='main_input')

 # Слой Embedding будет кодировать входную последовательность

#в последовательность 512-мерных векторов

x = Embedding(output_dim=512, input_dim=10000, input_length=100)(main_input)

 # LSTM преобразует векторную последовательность в один вектор,

 # содержащий информацию обо всей последовательности

lstm_out = LSTM(32)(x)

Здесь мы добавим вспомогательные ошибки, что позволит плавно обучать LSTM и E,bedding-слой, даже если ошибки основной модели будут намного выше.

auxiliary_output = Dense(1, activation='sigmoid', name='aux_output')(lstm_out)

На этом шаге мы вводим в модель дополнительные данные, объединяя их с выходом LSTM:

auxiliary_input = Input(shape=(5,), name='aux_input')

x = keras.layers.concatenate([lstm_out, auxiliary_input])

 # Вначале располагаем полносвязную сеть из 3 Dense-слоев

x = Dense(64, activation='relu')(x)

x = Dense(64, activation='relu')(x)

x = Dense(64, activation='relu')(x)

 # И в завершении добавляем основной слой логистической регрессии

main_output = Dense(1, activation='sigmoid', name='main_output')(x)

Определяем модель с двумя входами и двумя выходами:

model = Model(inputs=[main_input, auxiliary_input], outputs=[main_output, auxiliary_output])

Компилируем нашу модель и присваиваем вес 0.2 для значения вспомогательной ошибки. Для того чтобы указать разные loss_weights или loss для отдельного выхода, вы можете использовать список или словарь значений. Здесь мы передаем значение loss в качестве аргумента ошибки, поэтому одна и та же ошибка будет использоваться на всех выходах.

model.compile(optimizer='rmsprop',

loss={'main_output': 'binary_crossentropy', 'aux_output': 'binary_crossentropy'},

loss_weights={'main_output': 1., 'aux_output': 0.2})

 # Обучаем модель

model.fit({'main_input': headline_data, 'aux_input': additional_data},

{'main_output': headline_labels, 'aux_output': additional_labels},

epochs=50, batch_size=32)

Для вывода используйте:

model.predict({'main_input': headline_data, 'aux_input': additional_data})

или:

pred = model.predict([headline_data, additional_data])

Общие слои

Другое хорошее применение для функционального API — это модели, использующие общие слои. Давайте рассмотрим общие слои.

Рассмотрим набор данных твитов. Мы хотим построить модель, которая сможет определить, являются ли два твита от одного и того же человека или нет (это может позволить нам сравнить пользователей по сходству их твитов, например).

Одним из способов достижения этого является построение модели, которая кодирует два твита в два вектора, соединяет векторы, а затем добавляет логистическую регрессию; это выводит вероятность того, что два твита принадлежат одному и тому же автору. Затем модель будет обучена работе с положительными и отрицательными парами твитов.

Поскольку проблема симметрична, механизм кодирования первого твита следует использовать повторно (веса и все) для кодирования второго твита. Здесь мы используем общий LSTM слой для кодирования твитов.

Давайте построим это с помощью функционального API. Возьмем за вход для твита двоичную матрицу формы (280, 256), т.е. последовательность из 280 векторов размера 256, где каждое измерение в 256-мерном векторе кодирует наличие/отсутствие символа (из алфавита, состоящего из 256 частых символов).

import keras**from** keras.layers **import** Input, LSTM, Dense**from** keras.models **import** Model

tweet_a = Input(shape=(280, 256))

tweet_b = Input(shape=(280, 256))

Для совместного использования слоя с разными входными сигналами просто выполните однократную инициализацию слоя, а затем вызовите его на любое количество входных сигналов:

 # Этот слой может принять за вход матрицу # и вернет вектор размера 64

shared_lstm = LSTM(64)

*# Когда мы повторно используем один и тот же экземпляр слоя # несколько раз, вес слоя # также повторно используется # (это фактически *один и тот же * слой).*

encoded_a = shared_lstm(tweet_a)

encoded_b = shared_lstm(tweet_b)

 # Затем мы сможем соединить два вектора. #

```
merged_vector = keras.layers.concatenate([encoded_a, encoded_b], axis=-1)

# И добавить логистическую регрессию сверху.

predictions = Dense(1, activation='sigmoid')(merged_vector)

# Мы определяем обучаемую модель, связывающую входы tweet# с предсказаниями.

model = Model(inputs=[tweet_a, tweet_b], outputs=predictions)

model.compile(optimizer='rmsprop',

        loss='binary_crossentropy',

        metrics=['accuracy'])

model.fit([data_a, data_b], labels, epochs=10)
```

Давайте сделаем паузу, чтобы посмотреть, как читать выходную или выходную форму общего слоя.

Понятие «узел слоя»

При каждом вызове слоя на каком-либо входе создается новый тензор (выход слоя) и добавляется «узел» к слою, связывая входной тензор с выходным тензором. При многократном вызове одного и того же слоя этому слою принадлежат несколько узлов с индексами 0, 1, 2…

В предыдущих версиях Keras можно было получить выходной тензор экземпляра слоя с помощью layer.get_output() или его выходную форму с помощью layer.output_shape. И все же можно (за исключением того, что функция get_output() была заменена на выходное свойство). Но что делать, если слой подключен к нескольким входам?

Пока слой подключен только к одному входу, путаницы нет, и .output вернет один выход слоя:

```
a = Input(shape=(280, 256))

lstm = LSTM(32)
```

encoded_a = lstm(a)

assert lstm.output == encoded_a

Not so if the layer has multiple inputs:

a = Input(shape=(280, 256))

b = Input(shape=(280, 256))

lstm = LSTM(32)

encoded_a = lstm(a)

encoded_b = lstm(b)

lstm.output

>> AttributeError: Layer lstm_1 has multiple inbound nodes,

hence the notion **of** «layer output» **is** ill-defined.

Use `get_output_at(node_index)` instead.

Ну ладно. Следующее:

assert lstm.get_output_at(0) == encoded_a**assert** lstm.get_output_at(1) == encoded_b

Достаточно просто, да?

То же самое справедливо и для свойств input_shape и output_shape: до тех пор, пока у слоя есть только один узел или пока все узлы имеют одну и ту же форму входа/выхода, тогда понятие «форма выхода/входа слоя» четко определено, и одна форма будет возвращена слоем.output_shape/layer.input_shape. Но если, например, применить один и тот же слой Conv2D к входу формы (32, 32, 3), а затем к входу формы (64, 64, 3), то слой будет иметь несколько форм входа/выхода, и нужно будет получить их, указав индекс узла, к которому они принадлежат:

a = Input(shape=(32, 32, 3))

b = Input(shape=(64, 64, 3))

conv = Conv2D(16, (3, 3), padding='same')

conved_a = conv(a)

 # Пока только один вход, далее:

assert conv.input_shape == (**None**, 32, 32, 3)

conved_b = conv(b)*# теперь свойство `.input_shape` не сработает, но сработает следующее:*

assert conv.get_input_shape_at(0) == (**None**, 32, 32, 3)**assert** conv.get_input_shape_at(1) == (**None**, 64, 64, 3)

Другие примеры

Примеры кода все еще являются лучшим способом начать, так что вот еще несколько.

Начальный модуль

Для получения дополнительной информации об архитектуре Inception смотрите раздел Изучаем сверточные модели глубже.

from keras.layers **import** Conv2D, MaxPooling2D, Input

input_img = Input(shape=(256, 256, 3))

tower_1 = Conv2D(64, (1, 1), padding='same', activation='relu')(input_img)

tower_1 = Conv2D(64, (3, 3), padding='same', activation='relu')(tower_1)

tower_2 = Conv2D(64, (1, 1), padding='same', activation='relu')(input_img)

tower_2 = Conv2D(64, (5, 5), padding='same', activation='relu')(tower_2)

tower_3 = MaxPooling2D((3, 3), strides=(1, 1), padding='same')(input_img)

tower_3 = Conv2D(64, (1, 1), padding='same', activation='relu')(tower_3)

output = keras.layers.concatenate([tower_1, tower_2, tower_3], axis=1)

Остаточное соединение на сверточном слое

Дополнительные сведения об остаточных сетях см. в разделе Глубокое обучение остаточных сетей для распознавания изображений..

from keras.layers **import** Conv2D, Input

 # входной тензор для 3-канального изображения 256×256

x = Input(shape=(256, 256, 3))*# 3×3 conv с 3 выходными каналами (так же, как и входные каналы)*

y = Conv2D(3, (3, 3), padding='same')(x)*# возвращает x + y.*

z = keras.layers.add([x, y])

Модель общего видения

Эта модель использует один и тот же модуль обработки изображений на двух входах, чтобы классифицировать, являются ли две цифры MNIST одной и той же или разными цифрами.

from keras.layers **import** Conv2D, MaxPooling2D, Input, Dense, Flatten**from** keras.models **import** Model

 # Сначала определите модули видения

digit_input = Input(shape=(27, 27, 1))

x = Conv2D(64, (3, 3))(digit_input)

x = Conv2D(64, (3, 3))(x)

x = MaxPooling2D((2, 2))(x)

out = Flatten()(x)

vision_model = Model(digit_input, out)

 # Тогда определите модель tell-digits-apart.

digit_a = Input(shape=(27, 27, 1))

digit_b = Input(shape=(27, 27, 1))

 # Модель видения будет общей, веса и все все остальное.

out_a = vision_model(digit_a)

out_b = vision_model(digit_b)

concatenated = keras.layers.concatenate([out_a, out_b])

out = Dense(1, activation='sigmoid')(concatenated)

classification_model = Model([digit_a, digit_b], out)

Визуальная модель ответа на вопрос

Эта модель может выбрать правильный ответ из одного слова, когда задается естественный вопрос о картине.

Она работает, кодируя вопрос в вектор, кодируя изображение в вектор, объединяя их, и обучая сверху логистической регрессии по некоторому словарю потенциальных ответов.

from keras.layers **import** Conv2D, MaxPooling2D, Flatten**from** keras.layers **import** Input, LSTM, Embedding, Dense**from** keras.models **import** Model, Sequential

 # Сначала определим модель видения, используя Sequential model.# Эта модель будет кодировать изображение в вектор.

vision_model = Sequential()

vision_model.add(Conv2D(64, (3, 3), activation='relu', padding='same', input_shape=(224, 224, 3)))

vision_model.add(Conv2D(64, (3, 3), activation='relu'))

vision_model.add(MaxPooling2D((2, 2)))

vision_model.add(Conv2D(128, (3, 3), activation='relu', padding='same'))

vision_model.add(Conv2D(128, (3, 3), activation='relu'))

```
vision_model.add(MaxPooling2D((2, 2)))

vision_model.add(Conv2D(256, (3, 3), activation='relu', padding='same'))

vision_model.add(Conv2D(256, (3, 3), activation='relu'))

vision_model.add(Conv2D(256, (3, 3), activation='relu'))

vision_model.add(MaxPooling2D((2, 2)))

vision_model.add(Flatten())

# Теперь давайте возьмем тензор с выходом нашей модели видения:

image_input = Input(shape=(224, 224, 3))

encoded_image = vision_model(image_input)

# Далее определим языковую модель для кодирования вопроса в вектор.# Каждый вопрос будет длиной не более 100 слов,# и будем индексировать слова как целые числа от 1 до 9999.

question_input = Input(shape=(100,), dtype='int32')

embedded_question = Embedding(input_dim=10000, output_dim=256,
input_length=100)(question_input)

encoded_question = LSTM(256)(embedded_question)

# Соединим вектор вопроса и вектор изображения:

merged = keras.layers.concatenate([encoded_question, encoded_image])

# И давайте тренировать логистическую регрессию более чем на 1000 слов сверху:

output = Dense(1000, activation='softmax')(merged)

# Это наша последняя модель:

vqa_model = Model(inputs=[image_input, question_input], outputs=output)

# Следующим этапом будет обучение этой модели фактическим данным.
```

Видео модель ответа на вопрос

Теперь, когда мы обучили нашу модель контроля качества изображения, мы можем быстро превратить ее в модель контроля качества видео. При соответствующем тренинге вы сможете показать ему короткое видео (например, 100-кадровое человеческое действие) и задать естественный вопрос на языке видео (например, «в какой вид спорта играет мальчик?» -> «футбол»).

```
from keras.layers import TimeDistributed

video_input = Input(shape=(100, 224, 224, 3))# Это наше видео, закодированное с помощью ранее обученной модели vision_model (веса повторно используются).

encoded_frame_sequence = TimeDistributed(vision_model)(video_input)  # на выходе будет последовательность векторов

encoded_video = LSTM(256)(encoded_frame_sequence)  # на выходе будет векторный

# Это представление кодировщика вопросов на уровне модели с повторным использованием тех же весов, что и раньше:

question_encoder = Model(inputs=question_input, outputs=encoded_question)

# Давайте используем его для кодирования вопроса:

video_question_input = Input(shape=(100,), dtype='int32')

encoded_video_question = question_encoder(video_question_input)

# И это наша видео-модель, отвечающая на вопросы:

merged = keras.layers.concatenate([encoded_video, encoded_video_question])

output = Dense(1000, activation='softmax')(merged)

video_qa_model = Model(inputs=[video_input, video_question_input], outputs=output)
```