7 Лучшие практики глубокого обучения продвинутого уровня

Эта глава охватывает следующие темы:

- ✓ функциональный API фреймворка Keras;
- ✓ использование обратных вызовов Keras;
- ✓ использование инструмента визуализации TensorBoard;
- ✓ важнейшие лучшие практики для разработки современных моделей.

В этой главе рассматривается несколько мощных инструментов, позволяющих конструировать самые современные модели для решения сложных задач. Используя функциональный API фреймворка Keras, вы сможете строить графоподобные модели, повторно задействовать один и тот же слой для обработки разных входов и использовать модели Keras подобно функциям на языке Python. Обратные вызовы Keras и инструмент визуализации TensorBoard позволяют следить за процессом обучения моделей. Также мы обсудим некоторые другие продвинутые приемы, включая пакетную нормализацию, остаточные связи, оптимизацию гиперпараметров и ансамблирование.

7.1. За рамками модели Sequential: функциональный API фреймворка Keras

Все нейронные сети, представленные выше в этой книге, были реализованы с применением модели `Sequential`. Эта модель основана на предположении, что сеть имеет только один вход и только один выход и состоит из линейного стека слоев (рис. 7.1).

Это общепринятое предположение; данная конфигурация настолько распространена, что до настоящего момента мы смогли охватить множество тем и практических

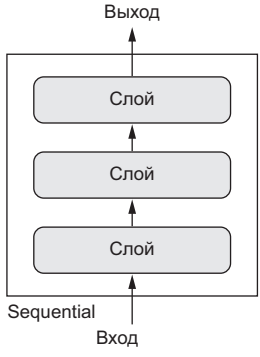

Рис. 7.1. Модель Sequential: линейный стек слоев

применений, использовав только класс `Sequential` моделей. Однако в ряде случаев это предположение оказывается слишком жестким. Некоторые сети требуют нескольких независимых входов, другие — нескольких выходов, а третьи имеют внутренние ветви, соединяющие слои, что делает их похожими на *графы*, а не на линейные стеки слоев.

Некоторые задачи, например, требуют мультимодальных входов: они объединяют данные, поступающие из разных источников, обрабатывая каждый вид данных с использованием разных типов нейронных слоев. Представьте модель глубокого обучения, которая пытается предсказать наиболее вероятную рыночную цену подержанной одежды, используя следующие входные данные: метаданные, представленные пользователем (такие, как торговая марка производителя, срок использования и т. д.), текстовое описание и фотографию. Если бы у вас имелись только метаданные, вы могли бы применить к ним метод прямого кодирования и использовать для предсказания цены полносвязную сеть. Если бы у вас имелось только текстовое описание, вы могли бы использовать рекуррентную или одномерную сверточную сеть. Если бы у вас имелась только фотография, вы могли бы использовать двумерную сверточную сеть. Но как использовать все три вида входных данных одновременно? Простейшим решением могло бы быть обучение трех отдельных моделей с последующим вычислением взвешенного среднего их предсказаний. Однако такое решение может оказаться не самым оптимальным, потому что информация, извлекаемая моделями, может быть избыточной. Лучшим решением является изучение более точной модели данных с использованием модели машинного обучения, которая способна обрабатывать все входные модальности одновременно, — модели с тремя входными ветвями (рис. 7.2).

Аналогично, в некоторых задачах требуется предсказать несколько целевых атрибутов по входным данным. Например, по тексту повести или рассказа определить жанр (любовный роман или военная повесть) и примерную дату их написания. Конечно, можно подготовить две отдельные модели: одну для определения жанра, а другую — для даты. Однако поскольку эти атрибуты не являются статистически независимыми, лучшим решением будет создать модель, обучающуюся для одновременного

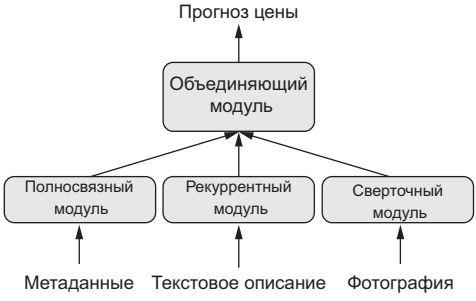

Рис. 7.2. Модель с несколькими входами

предсказания обоих выходных атрибутов — жанра и даты. Такая объединенная модель будет иметь два выхода, или *головы* (рис. 7.3). Благодаря корреляции между жанром и датой, знание даты может помочь модели получить богатое и точное представление пространства жанров литературных произведений, и наоборот.

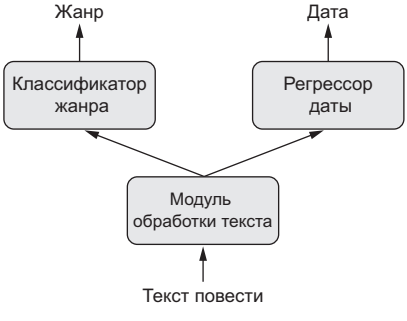

Рис. 7.3. Модель с несколькими выходами (головами)

Кроме того, многие нейронные архитектуры, разработанные недавно, требуют нелинейной организации сети, когда сеть имеет структуру ориентированного ациклического графа. Семейство сетей Inception (разработанное Кристианом Сегеди (Christian Szegedy) с коллегами в Google)[1], например, опирается на *модули Inception*, в которых входные данные обрабатываются несколькими параллельными сверточными ветвями, выходы которых затем объединяются в единый тензор (рис. 7.4). Также недавно появилась методика добавления в модель *остаточных связей*, развитие которой началось с появления семейства сетей ResNet (разработанного Каймином Хе (Kaiming He) с коллегами в Microsoft)[2]. Суть этого приема заключается в повторном внедрении предыдущих представлений в исходящий поток данных добавлением про-

[1] Christian Szegedy et al., «Going Deeper with Convolutions», Conference on Computer Vision and Pattern Recognition (2014), *https://arxiv.org/abs/1409.4842*.

[2] Kaiming He et al., «Deep Residual Learning for Image Recognition», Conference on Computer Vision and Pattern Recognition (2015), *https://arxiv.org/abs/1512.03385*.

шлого выходного тензора к более новому выходному тензору (рис. 7.5), что помогает предотвратить потерю информации в процессе обработки данных. Существует также множество других примеров, таких как графоподобные сети.

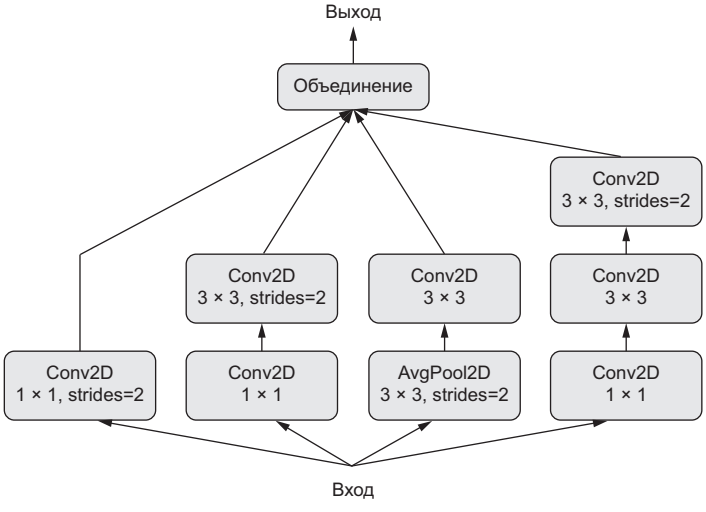

Рис. 7.4. Модуль Inception: подграф уровней с несколькими параллельными сверточными ветвями

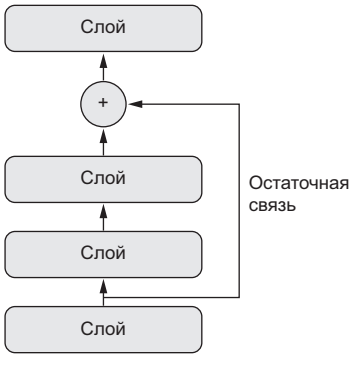

Рис. 7.5. Остаточные связи: повторное внедрение предыдущей исходящей информации добавлением в карту признаков

Эти три важные разновидности моделей — модели с несколькими входами, модели с несколькими выходами и графоподобные модели — невозможно реализовать с использованием только класса `Sequential` моделей из фреймворка Keras. Однако существует другой, намного более универсальный и гибкий способ использования Keras — *функциональный API*. В этом разделе подробно рассказывается, что это такое, описываются его возможности и особенности использования.

7.1.1. Введение в функциональный API

Функциональный API позволяет напрямую манипулировать тензорами и использовать уровни как функции, которые принимают и возвращают тензоры (чем и обусловлено такое название — *функциональный API*):

```
from keras import Input, layers

input_tensor = Input(shape=(32,))  <-- Тензор

dense = layers.Dense(32, activation='relu')  <-- Слой — это функция

output_tensor = dense(input_tensor)  <-- Вызываемый слой может принимать и возвращать тензор
```

Начнем с маленького примера, демонстрирующего простую модель `Sequential` и ее эквивалент с использованием функционального API:

```
from keras.models import Sequential, Model
from keras import layers
from keras import Input

seq_model = Sequential()  <-- Уже знакомая нам модель Sequential
seq_model.add(layers.Dense(32, activation='relu', input_shape=(64,)))
seq_model.add(layers.Dense(32, activation='relu'))
seq_model.add(layers.Dense(10, activation='softmax'))

input_tensor = Input(shape=(64,))
x = layers.Dense(32, activation='relu')(input_tensor)
x = layers.Dense(32, activation='relu')(x)
output_tensor = layers.Dense(10, activation='softmax')(x)
```

Ее функциональный эквивалент

```
model = Model(input_tensor, output_tensor)  <-- Класс Model превращает входной и выходной тензоры в модель

model.summary()  <-- Рассмотрим ее!
```

Вот что вывел вызов `model.summary()`:

```
_________________________________________________________________
Layer (type) Output Shape Param #
=================================================================
input_1 (InputLayer)            (None, 64)            0
_________________________________________________________________
dense_1 (Dense)                 (None, 32)            2080
_________________________________________________________________
dense_2 (Dense)                 (None, 32)            1056
_________________________________________________________________
dense_3 (Dense)                 (None, 10)            330
=================================================================
Total params: 3,466
Trainable params: 3,466
Non-trainable params: 0
_________________________________________________________________
```

Единственное, что может показаться здесь немного необычным, — это создание экземпляра класса `Model` только с использованием входного и выходного тензоров. За кулисами Keras извлекает все слои, участвовавшие в преобразовании тензора `input_tensor` в тензор `output_tensor`, и объединяет их в графоподобную структуру данных — `Model`. Конечно, подобное возможно только для выходного тензора `output_tensor`, полученного путем многократных преобразований входного тензора `input_tensor`. Если попытаться создать модель из не связанных между собой входов и выходов, вы получите исключение `RuntimeError`:

```
>>> unrelated_input = Input(shape=(32,))
>>> bad_model = model = Model(unrelated_input, output_tensor)
RuntimeError: Graph disconnected: cannot
obtain value for tensor
➥Tensor("input_1:0", shape=(?, 64), dtype=float32) at layer "input_1".
```

Это исключение фактически сообщает, что фреймворку Keras не удалось достичь `input_1` из переданного ему выходного тензора.

Компиляция, обучение и оценка такого экземпляра `Model` выглядит точно так же, как при использовании модели `Sequential`:

```
model.compile(optimizer='rmsprop', loss='categorical_crossentropy')

import numpy as np
x_train = np.random.random((1000, 64))
y_train = np.random.random((1000, 10))

model.fit(x_train, y_train, epochs=10, batch_size=128)

score = model.evaluate(x_train, y_train)
```

Компиляция модели

Генерация фиктивных данных для обучения

Обучение модели на протяжении 10 эпох

Оценка модели

7.1.2. Модели с несколькими входами

Функциональный API можно использовать для создания моделей с несколькими входами. Обычно такие модели в какой-то момент объединяют свои входные ветви, используя слой, способный объединить несколько тензоров: сложением, слиянием или как-то иначе. Часто для этого используются операции слияния, реализованные в Keras, такие как `keras.layers.add`, `keras.layers.concatenate` и т. д. Рассмотрим пример очень простой модели с несколькими входами: модель вида «вопрос/ответ».

Типичная модель «вопрос/ответ» имеет два входа: вопрос на естественном языке и фрагмент текста (например, новостная статья) с информацией, которая будет использоваться для ответа на вопрос. Опираясь на эти данные, модель должна вернуть ответ: в простейшем случае это может быть ответ, состоящий из одного слова, полученного применением классификатора `softmax` к некоторому предопределенному словарю (рис. 7.6).

Следующий пример демонстрирует, как создать модель с помощью функционального API. В нем создаются две независимые ветви, входной текст и вопрос коди-

руются в векторные представления; затем эти векторы объединяются и, наконец, поверх объединенного представления добавляется классификатор `softmax`.

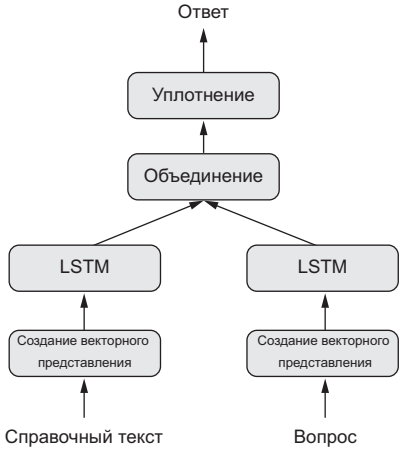

Рис. 7.6. Модель «вопрос/ответ»

Листинг 7.1. Реализация модели «вопрос/ответ» с двумя входами с использованием функционального API

```
from keras.models import Model
from keras import layers
from keras import Input

text_vocabulary_size = 10000
question_vocabulary_size = 10000
answer_vocabulary_size = 500

text_input = Input(shape=(None,), dtype='int32', name='text')

embedded_text = layers.Embedding(
    text_vocabulary_size, 64)(text_input)

encoded_text = layers.LSTM(32)(embedded_text)

question_input = Input(shape=(None,),
                       dtype='int32',
                       name='question')

embedded_question = layers.Embedding(
    question_vocabulary_size, 32)(question_input)
encoded_question = layers.LSTM(16)(embedded_question)

concatenated = layers.concatenate([encoded_text, encoded_question],
                                  axis=-1)
```

Входной текст — это последовательность целых чисел переменной длины. Обратите внимание на то, что при желании можно задать имя последовательности

Преобразование входного текста в последовательность векторов с размером 64

Преобразование векторов в единый вектор с помощью уровня LSTM

Та же процедура (с другими экземплярами слоев) повторяется для вопроса

Объединение закодированных вопроса и текста

```
answer = layers.Dense(answer_vocabulary_size,
                      activation='softmax')(concatenated)

model = Model([text_input, question_input], answer)
model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['acc'])
```

Создание экземпляра модели с двумя входами и одним выходом

Добавление классификатора softmax сверху

Теперь встает вопрос обучения модели с двумя входами. Для этого можно использовать два разных API: можно передать модели список массивов Numpy или словарь, отображающий имена входов в массивы Numpy. Естественно, последний вариант возможен, только если вы определили имена для входов.

Листинг 7.2. Передача данных в модель с несколькими входами

```
import numpy as np

num_samples = 1000
max_length = 100

text = np.random.randint(1, text_vocabulary_size,
                         size=(num_samples, max_length))
question = np.random.randint(1, question_vocabulary_size,
                             size=(num_samples, max_length))

answers = np.zeros(shape=(num_samples, answer_vocabulary_size))
indices = np.random.randint(0, answer_vocabulary_size, size=num_samples)
for i, x in enumerate(answers):
    x[indices[i]] = 1

model.fit([text, question], answers, epochs=10, batch_size=128)

model.fit({'text': text, 'question': question}, answers,
          epochs=10, batch_size=128)
```

Создание массива Numpy с фиктивными данными

К вопросам применяется прямое кодирование, а не преобразование в целые числа

Передача списка входов

Передача с помощью словаря (возможна, только если были определены имена для входов)

7.1.3. Модели с несколькими выходами

Функциональный API также можно использовать для создания моделей с несколькими выходами (или *головами*). Простейшим примером может служить сеть, пытающаяся одновременно предсказать разные свойства данных, например принимающая на входе последовательность постов из социальной сети от некоторой анонимной персоны и пытающаяся предсказать характеристики этой персоны, такие как возраст, пол и уровень доходов (рис. 7.7).