БИБЛИОТЕКИ ЯЗЫКА PYTHON ДЛЯ КОМПЬЮТЕРНЫХ ВЫЧИСЛЕНИЙ И МОДЕЛИРОВАНИЯ

Библиотеки **Python** для машинного обучение (Ч. 2)

Лобанов Алексей Владимирович
Главный специалист отдела разработки и внедрения АИС,
Ассистент кафедры информационных компьютерных технологий РХТУ им. Д.И. Менделеева

ПОЛЕЗНЫЕ РЕСУРСЫ

- Библиотека TensorFlow
- Библиотека Keras
- Библиотека Theano
- Библиотека Scikit-learn
- Библиотека PyTorch
- Библиотека NumPy
- Библиотека Pandas

ТЕМЫ

- Классификация и регрессия
- Основы машинного обучения

КЛАССИФИКАЦИЯ

Пример для классификации новостных лент агентства Reuters на 46 взаимоисключающих тем. Так как теперь количество классов больше двух, эта задача относится к категории задач многоклассовой классификации; и, поскольку каждый экземпляр данных должен быть отнесен только к одному классу, эта задача является примером однозначной многоклассовой классификации. Если бы каждый экземпляр данных мог принадлежать нескольким классам (в данном случае темам), эта задача была бы примером многозначной многоклассовой классификации.

Листинг 3.12. Загрузка данных Reuters

```
from keras.datasets import reuters

(train_data, train_labels), (test_data, test_labels) = reuters.load_data(
    num_words=10000)
```

По аналогии с примером IMDB, аргумент `num_words=10000` ограничивает данные 10 000 наиболее часто встречающимися словами.

Всего у нас имеется 8982 обучающих и 2246 контрольных примеров:

```
>>> len(train_data)
8982
>>> len(test_data)
2246
```

По аналогии с отзывами в базе данных IMDB, каждый пример — это список целых чисел (индексов слов):

```
>>> train_data[10]
[1, 245, 273, 207, 156, 53, 74, 160, 26, 14, 46, 296, 26, 39, 74, 2979,
3554, 14, 46, 4689, 4329, 86, 61, 3499, 4795, 14, 61, 451, 4329, 17, 12]
```

Метка, определяющая класс примера, — это целое число между 0 и 45 — индекс темы:

```
>>> train_labels[10]
3
```

КЛАССИФИКАЦИЯ

Листинг 3.14. Кодирование данных

```
import numpy as np

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

x_train = vectorize_sequences(train_data)
x_test = vectorize_sequences(test_data)
```

← Векторизованные обучающие данные

← Векторизованные контрольные данные

Векторизовать метки можно одним из двух способов: сохранить их в тензоре целых чисел или использовать прямое кодирование. Прямое кодирование (one-hot encoding) широко используется для форматирования категорий и также называется *кодированием категорий* (categorical encoding).

Следует отметить, что этот способ уже реализован в Keras, как мы видели в примере MNIST:

```
from keras.utils.np_utils import to_categorical

one_hot_train_labels = to_categorical(train_labels)
one_hot_test_labels = to_categorical(test_labels)
```

← Векторизованные обучающие данные

← Векторизованные контрольные данные

Конструирование сети

Задача классификации по темам напоминает предыдущую задачу классификации отзывов: в обоих случаях мы пытаемся классифицировать короткие фрагменты текста. Но в данном случае количество выходных классов увеличилось с 2 до 46. Размерность выходного пространства теперь намного больше.

В стеке слоев `Dense`, как в предыдущем примере, каждый слой имеет доступ только к информации, предоставленной предыдущим слоем. Если один слой отбросит какую-то информацию, важную для решения задачи классификации, последующие слои не смогут восстановить ее: каждый слой может стать узким местом для информации. В предыдущем примере мы использовали 16-мерные промежуточные слои, но 16-мерное пространство может оказаться слишком ограниченным для классификации на 46 разных классов: такие малоразмерные слои могут сыграть роль «бутылочного горлышка» для информации, не пропуская важные данные.

По этой причине в данном примере мы будем использовать слои с большим количеством измерений. Давайте выберем 64 нейрона.

Листинг 3.15. Определение модели

```
from keras import models
from keras import layers

model = models.Sequential()
model.add(layers.Dense(64, activation='relu', input_shape=(10000,)))
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(46, activation='softmax'))
```

Листинг 3.16. Компиляция модели

```
model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

КЛАССИФИКАЦИЯ

Листинг 3.17. Создание проверочного набора

```
x_val = x_train[:1000]
partial_x_train = x_train[1000:]

y_val = one_hot_train_labels[:1000]
partial_y_train = one_hot_train_labels[1000:]
```

Теперь проведем обучение модели в течение 20 эпох.

Листинг 3.18. Обучение модели

```
history = model.fit(partial_x_train,
                    partial_y_train,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_val, y_val))
```

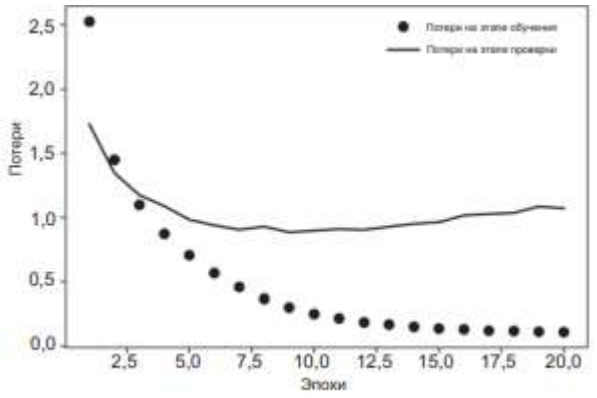
И наконец, выведем графики кривых потерь и точности (рис. 3.9 и 3.10).

Листинг 3.19. Формирование графиков потерь на этапах обучения и проверки

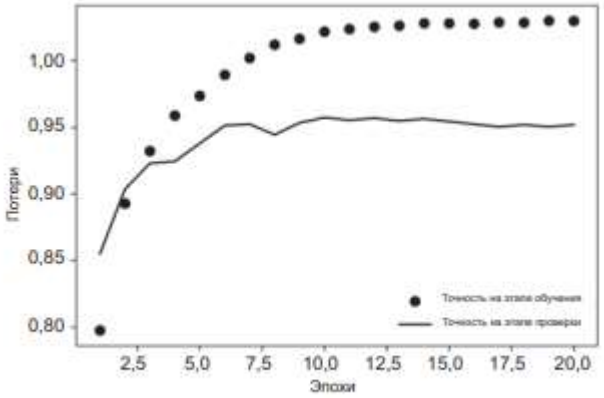
```
import matplotlib.pyplot as plt

loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(1, len(loss) + 1)

plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.show()
```

Листинг 3.20. Формирование графиков точности на этапах обучения и проверки

```
plt.clf()   ← Очистить рисунок

acc = history.history['acc']
val_acc = history.history['val_acc']

plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

plt.show()
```



Потери на этапах обучения и проверки



Точность на этапах обучения и проверки

КЛАССИФИКАЦИЯ

Переобучение сети наступает в девятой эпохе. Давайте теперь обучим новую сеть до девятой эпохи и затем оценим получившийся результат на контрольных данных.

Листинг 3.21. Обучение новой модели с нуля

```
model = models.Sequential()
model.add(layers.Dense(64, activation='relu', input_shape=(10000,)))
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(46, activation='softmax'))

model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

model.fit(partial_x_train,
          partial_y_train,
          epochs=9,
          batch_size=512,
          validation_data=(x_val, y_val))

results = model.evaluate(x_test, one_hot_test_labels)
```

Конечные результаты:

```
>>> results
[0.9565213431445807, 0.79697239536954589]
```

Это решение достигло точности ~80 %. Со сбалансированной задачей бинарной классификации точность чисто случайного классификатора составила бы 50 %. Однако в данном случае она близка к 19 %, то есть результат получился весьма неплохим, по крайней мере в сравнении со случайным решением:

```
>>> import copy
>>> test_labels_copy = copy.copy(test_labels)
>>> np.random.shuffle(test_labels_copy)
>>> hits_array = np.array(test_labels) == np.array(test_labels_copy)
>>> float(np.sum(hits_array)) / len(test_labels)
0.18655387355298308
```

КЛАССИФИКАЦИЯ

Предсказания на новых данных

Теперь можно убедиться в том, что метод `predict` модели возвращает распределение вероятностей по всем 46 темам. Давайте сгенерируем предсказания для всех контрольных данных.

Листинг 3.22. Получение предсказаний для новых данных

```
predictions = model.predict(x_test)
```

Каждый элемент в `predictions` — это вектор с длиной 46:

```
>>> predictions[0].shape
(46,)
```

Сумма коэффициентов этого вектора равна 1:

```
>>> np.sum(predictions[0])
1.0
```

Наибольший элемент, элемент с наибольшей вероятностью, — это предсказанный класс:

```
>>> np.argmax(predictions[0])
4
```

Другой способ обработки меток и потерь

Выше упоминалось, что метки также можно было бы преобразовать в тензор целых чисел, как показано ниже:

```
y_train = np.array(train_labels)
y_test = np.array(test_labels)
```

Единственное, что изменилось в данном случае, — функция потерь. В листинге 3.21 использовалась функция потерь `categorical_crossentropy`, предполагающая, что метки получены методом кодирования категорий. С целочисленными метками следует использовать функцию `sparse_categorical_crossentropy`:

```
model.compile(optimizer='rmsprop',
              loss='sparse_categorical_crossentropy',
              metrics=['acc'])
```

С математической точки зрения эта новая функция потерь равноценна функции `categorical_crossentropy`; ее отличает только интерфейс.

КЛАССИФИКАЦИЯ

Важность использования достаточно больших промежуточных слоев

Выше уже говорилось, что не следует использовать слои, в которых значительно меньше 46 скрытых нейронов, потому что результат является 46-мерным. Теперь давайте посмотрим, что получится, если образуется узкое место для информации из-за промежуточных слоев с размерностями намного меньше 46, например четырехмерных.

Теперь сеть показывает точность ~71 % — абсолютное падение составило 8 %. Это падение в основном обусловлено попыткой сжать большой объем информации (достаточной для восстановления гиперплоскостей, разделяющих 46 классов) в промежуточное пространство со слишком малой размерностью. Сети удалось вместить *бóльшую* часть необходимой информации в эти четырехмерные представления, но не всю.

Листинг 3.23. Модель с узким местом для информации

```
model = models.Sequential()
model.add(layers.Dense(64, activation='relu', input_shape=(10000,)))
model.add(layers.Dense(4, activation='relu'))
model.add(layers.Dense(46, activation='softmax'))

model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

model.fit(partial_x_train,
          partial_y_train,
          epochs=20,
          batch_size=128,
          validation_data=(x_val, y_val))
```

Подведение итогов

Вот какие выводы вы должны сделать из этого примера:

- Если вы пытаетесь классифицировать образцы данных по *N* классам, сеть должна завершаться слоем `Dense` размера *N*.
- В задаче однозначной многоклассовой классификации заключительный слой сети должен иметь функцию активации `softmax`, чтобы он мог выводить распределение вероятностей между *N* классами.
- Для решения подобных задач почти всегда следует использовать функцию потерь `categorical_crossentropy`. Она минимизирует расстояние между распределениями вероятностей, выводимыми сетью, и истинными распределениями целей.
- Метки в многоклассовой классификации можно обрабатывать двумя способами:
  - Кодировать метки с применением метода кодирования категорий (также известного как прямое кодирование) и использовать функцию потерь `categorical_crossentropy`.
  - Кодировать метки как целые числа и использовать функцию потерь `sparse_categorical_crossentropy`.
- Когда требуется классифицировать данные относительно большого количества категорий, следует предотвращать появление в сети узких мест для информации из-за слоев с недостаточно большим количеством измерений.

РЕГРЕССИЯ

Набор данных с ценами на жилье в Бостоне

Мы попытаемся предсказать медианную цену на дома в пригороде Бостона в середине 1970-х по таким данным о пригороде того времени, как уровень преступности, ставка местного имущественного налога и т. д. Набор данных, который нам предстоит использовать, имеет интересное отличие от двух предыдущих примеров. Он содержит относительно немного образцов данных: всего 506, разбитых на 404 обучающих и 102 контрольных образца. И каждый *признак* во входных данных (например, уровень преступности) имеет свой масштаб. Например, некоторые признаки являются пропорциями и имеют значения между 0 и 1, другие — между 1 и 12 и т. д.

Листинг 3.24. Загрузка набора данных для Бостона

```
from keras.datasets import boston_housing

(train_data, train_targets), (test_data, test_targets) =
➥boston_housing.load_data()
```

Посмотрим на данные:

```
>>> train_data.shape
(404, 13)
>>> test_data.shape
(102, 13)
```

Как видите, у нас имеются 404 обучающих и 102 контрольных образца, каждый с 13 числовыми признаками, такими как уровень преступности, среднее число комнат в доме, удаленность от центральных дорог и т. д.

Цели — медианные значения цен на дома, занимаемые собственниками, в тысячах долларов:

```
>>> train_targets
[ 15.2, 42.3, 50. ... 19.4, 19.4, 29.1]
```

Цены в основной массе находятся в диапазоне от 10 000 до 50 000 долларов США. Если вам покажется, что это недорого, не забывайте, что это цены середины 1970-х и в них не были внесены поправки на инфляцию.

Подготовка данных

Было бы проблематично передать в нейронную сеть значения, имеющие самые разные диапазоны. Сеть, конечно, сможет автоматически адаптироваться к таким разнородным данным, однако это усложнит обучение. На практике к таким данным принято применять нормализацию: для каждого признака во входных данных (столбца в матрице входных данных) из каждого значения вычитается среднее по этому признаку, и разность делится на стандартное отклонение, в результате признак центрируется по нулевому значению и имеет стандартное отклонение, равное единице. Такую нормализацию легко выполнить с помощью Numpy.

Листинг 3.25. Нормализация данных

```
mean = train_data.mean(axis=0)
train_data -= mean
std = train_data.std(axis=0)
train_data /= std

test_data -= mean
test_data /= std
```

Обратите внимание на то, что величины, используемые для нормализации контрольных данных, вычисляются с использованием обучающих данных. Никогда не следует использовать в работе какие-либо значения, вычисленные по контрольным данным, даже для таких простых шагов, как нормализация данных.

РЕГРЕССИЯ

Конструирование сети

Из-за небольшого количества образцов мы будем использовать очень маленькую сеть с двумя четырехмерными промежуточными слоями. Вообще говоря, чем меньше обучающих данных, тем скорее наступит переобучение, а использование маленькой сети — один из способов борьбы с ним.

Листинг 3.26. Определение модели

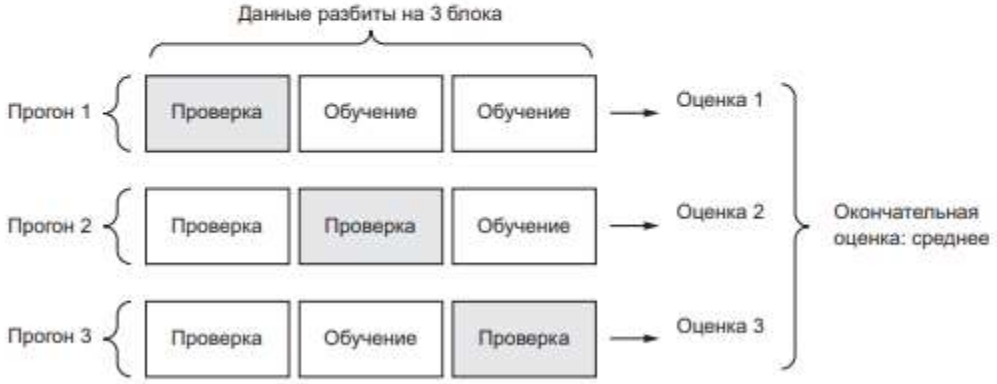
```
from keras import models
from keras import layers

def build_model():
    model = models.Sequential()
    model.add(layers.Dense(64, activation='relu',
                           input_shape=(train_data.shape[1],)))
    model.add(layers.Dense(64, activation='relu'))
    model.add(layers.Dense(1))
    model.compile(optimizer='rmsprop', loss='mse', metrics=['mae'])
    return model
```

Поскольку нам потребуется несколько экземпляров одной и той же модели, мы определили функцию для ее создания

Оценка решения методом перекрестной проверки по K блокам

Чтобы оценить качество сети в ходе корректировки ее параметров (таких, как количество эпох обучения), можно разбить исходные данные на обучающий и проверочный наборы, как это делалось в предыдущих примерах. Однако так как у нас и без того небольшой набор данных, проверочный набор получился бы слишком маленьким (скажем, что-нибудь около 100 образцов). Как следствие, оценки при проверке могут сильно меняться в зависимости от того, какие данные попадут в проверочный и обучающий наборы: оценки при проверке могут иметь слишком большой разброс. Это не позволит надежно оценить качество модели.

Лучшей практикой в таких ситуациях является применение *перекрестной проверки по K блокам* (K-fold cross-validation), как показано на рис. . Суть ее заключается в разделении доступных данных на K блоков (обычно K = 4 или 5), создании K идентичных моделей и обучении каждой на K−1 блоках с оценкой по оставшимся блокам. По полученным K оценкам вычисляется среднее значение, которое принимается как оценка модели. В коде такая проверка реализуется достаточно просто.



Перекрестная проверка по трем блокам

РЕГРЕССИЯ

Листинг 3.27. Перекрестная проверка по K блокам

```
import numpy as np

k = 4
num_val_samples = len(train_data) // k
num_epochs = 100
all_scores = []

for i in range(k):
    print('processing fold #', i)
    val_data = train_data[i * num_val_samples: (i + 1) * num_val_samples]
    val_targets = train_targets[i * num_val_samples: (i + 1)
                  * num_val_samples]

    partial_train_data = np.concatenate(
        [train_data[:i * num_val_samples],
         train_data[(i + 1) * num_val_samples:]],
        axis=0)
    partial_train_targets = np.concatenate(
        [train_targets[:i * num_val_samples],
         train_targets[(i + 1) * num_val_samples:]],
        axis=0)

    model = build_model()

    model.fit(partial_train_data, partial_train_targets,
              epochs=num_epochs, batch_size=1, verbose=0)
    val_mse, val_mae = model.evaluate(val_data, val_targets, verbose=0)
    all_scores.append(val_mae)
```

Подготовка проверочных данных: данных из блока с номером k

Подготовка обучающих данных: данных из остальных блоков

Конструирование модели Keras (уже скомпилированной)

Обучение модели (в режиме без вывода сообщений, verbose = 0)

Оценка модели по проверочным данным

Выполнив этот код с `num_epochs = 100`, мы получили следующие результаты:

```
>>> all_scores
[2.588258957792037, 3.1289568449719116, 3.1856116051248984, 3.0763342615401386]
>>> np.mean(all_scores)
2.9947904173572462
```

Разные прогоны действительно показывают разные оценки, от 2,6 до 3,2. Средняя (3,0) выглядит более достоверно, чем любая из оценок отдельных прогонов, — в этом главная ценность перекрестной проверки по K блокам. В данном случае средняя ошибка составила 3000 долларов, что довольно много, если вспомнить, что цены колеблются в диапазоне от 10 000 до 50 000 долларов.

Листинг 3.28. Сохранение оценки проверки перед каждым прогоном

```
num_epochs = 500
all_mae_histories = []
for i in range(k):
    print('processing fold #', i)
    val_data = train_data[i * num_val_samples: (i + 1) * num_val_samples]
    val_targets = train_targets[i * num_val_samples:
                  (i + 1) * num_val_samples]

    partial_train_data = np.concatenate(
        [train_data[:i * num_val_samples],
         train_data[(i + 1) * num_val_samples:]],
        axis=0)
    partial_train_targets = np.concatenate(
        [train_targets[:i * num_val_samples],
         train_targets[(i + 1) * num_val_samples:]],
        axis=0)

    model = build_model()
    history = model.fit(partial_train_data, partial_train_targets,
                        validation_data=(val_data, val_targets),
                        epochs=num_epochs, batch_size=1, verbose=0)
    mae_history = history.history['val_mean_absolute_error']
    all_mae_histories.append(mae_history)
```

Подготовка проверочных данных: данных из блока с номером k

Подготовка обучающих данных: данных из остальных блоков

Конструирование модели Keras (уже скомпилированной)

Обучение модели (в режиме без вывода сообщений, verbose = 0)

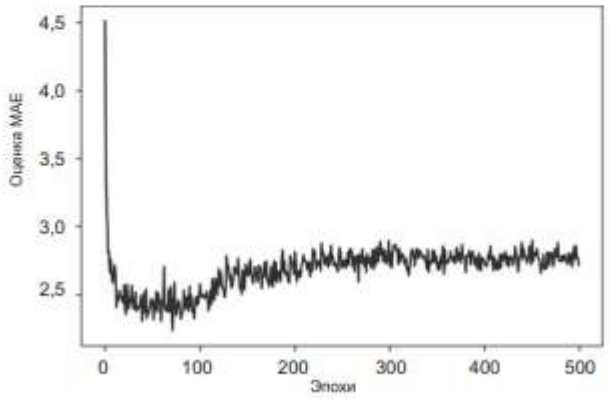
РЕГРЕССИЯ

Листинг 3.29. Создание истории последовательных средних оценок проверки по K блокам

```
average_mae_history = [
    np.mean([x[i] for x in all_mae_histories]) for i in range(num_epochs)]
```

Листинг 3.30. Формирование графика с оценками проверок

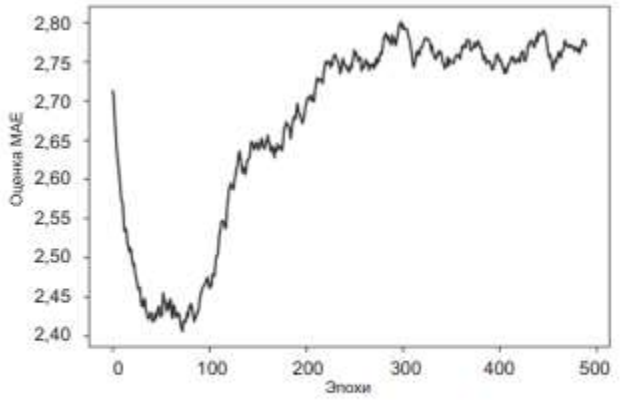
```
import matplotlib.pyplot as plt

plt.plot(range(1, len(average_mae_history) + 1), average_mae_history)
plt.xlabel('Epochs')
plt.ylabel('Validation MAE')
plt.show()
```



Оценки MAE по эпохам

Листинг 3.31. Формирование графика с оценками проверок за исключением первых 10 замеров

```
def smooth_curve(points, factor=0.9):
    smoothed_points = []
    for point in points:
        if smoothed_points:
            previous = smoothed_points[-1]
            smoothed_points.append(previous * factor + point * (1 - factor))
        else:
            smoothed_points.append(point)
    return smoothed_points

smooth_mae_history = smooth_curve(average_mae_history[10:])

plt.plot(range(1, len(smooth_mae_history) + 1), smooth_mae_history)
plt.xlabel('Epochs')
plt.ylabel('Validation MAE')
plt.show()
```



Оценки MAE по эпохам за исключением первых 10 замеров

РЕГРЕССИЯ

Листинг 3.32. Обучение окончательной версии модели

```
model = build_model()
model.fit(train_data, train_targets,
          epochs=80, batch_size=16, verbose=0)
test_mse_score, test_mae_score = model.evaluate(test_data, test_targets)
```

Получить новую скомпилированную модель

Обучить ее на всем объеме обучающих данных

Вот окончательный результат:

```
>>> test_mae_score
2.5532484335057877
```

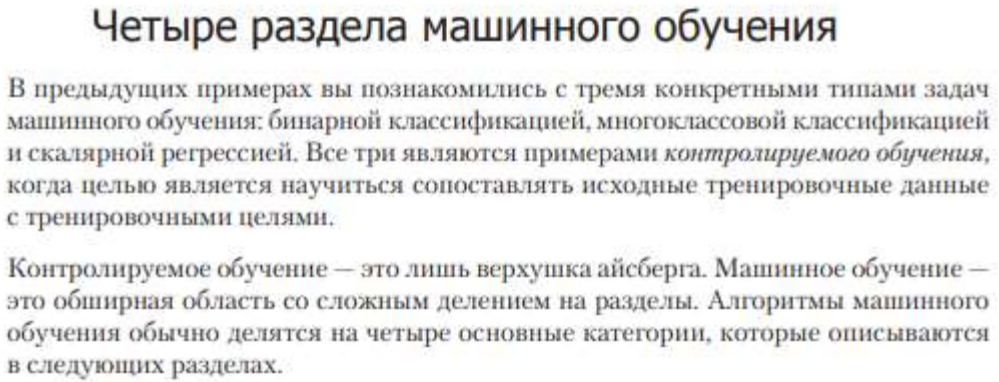
Средняя ошибка все еще составляет около 2550 долларов.

Подведение итогов

Вот какие выводы вы должны сделать из этого примера:

- Регрессия выполняется с применением иных функций потерь, нежели классификация. Для регрессии часто используется функция потерь, вычисляющая среднеквадратичную ошибку (Mean Squared Error, MSE).
- Аналогично, для регрессии используются иные метрики оценки, нежели при классификации; понятие точности неприменимо для регрессии, поэтому для оценки качества часто применяется средняя абсолютная ошибка (Mean Absolute Error, MAE).
- Когда признаки образцов на входе имеют значения из разных диапазонов, их необходимо предварительно масштабировать.
- При небольшом объеме входных данных надежно оценить качество модели поможет метод перекрестной проверки по K блокам.
- При небольшом объеме обучающих данных предпочтительнее использовать маленькие сети с небольшим количеством промежуточных слоев (обычно с одним или двумя), чтобы избежать серьезного переобучения.

МАШИННОЕ ОБУЧЕНИЕ



Четыре раздела машинного обучения

В предыдущих примерах вы познакомились с тремя конкретными типами задач машинного обучения: бинарной классификацией, многоклассовой классификацией и скалярной регрессией. Все три являются примерами *контролируемого обучения*, когда целью является научиться сопоставлять исходные тренировочные данные с тренировочными целями.

Контролируемое обучение — это лишь верхушка айсберга. Машинное обучение — это обширная область со сложным делением на разделы. Алгоритмы машинного обучения обычно делятся на четыре основные категории, которые описываются в следующих разделах.

В предыдущих примерах вы познакомились с двумя конкретными типами задач машинного обучения: бинарной классификацией, многоклассовой классификацией. Все три являются примерами контролируемого обучения, когда целью является научиться сопоставлять исходные тренировочные данные с тренировочными целями.

Четыре раздела машинного обучения

- Контролируемое обучение
- Неконтролируемое обучение
- Самоконтролируемое обучение
- Обучение с подкреплением

МАШИННОЕ ОБУЧЕНИЕ

Контролируемое обучение

Этот случай, безусловно, является наиболее распространенным. Его суть заключается в том, чтобы научить модель отображать исходные данные в известные целевые значения (иногда их также называют *аннотациями*) на наборе примеров (часто аннотированных людьми). Все четыре примера, которые вы видели выше в этой книге, являются каноническими примерами контролируемого обучения. Вообще говоря, к этой категории относятся почти все современные способы применения глубокого обучения, такие как распознавание образов, распознавание речи, классификация изображений и перевод с одного языка на другой.

Неконтролируемое обучение

Этот раздел машинного обучения заключается в поиске интересных преобразований входных данных без помощи каких-либо целевых значений для нужд визуализации, сжатия или очистки данных от шумов или для лучшего понимания взаимосвязей в данных. Неконтролируемое обучение — это основа анализа данных, оно часто оказывается необходимым шагом на пути изучения набора данных перед применением методов контролируемого обучения. Хорошо известными примерами неконтролируемого обучения являются *понижение размерности* и *кластеризация*.

Самоконтролируемое обучение

Это разновидность контролируемого обучения, которая имеет существенные отличия, а потому может быть выделена в отдельную категорию. Самоконтролируемое обучение контролируется без использования меток, расставленных человеком, — самоконтролируемое обучение можно считать контролируемым обучением без участия людей. Здесь также имеются метки (иначе это обучение не было бы контролируемым), однако они генерируются из исходных данных, обычно с применением эвристических алгоритмов.

Обучение с подкреплением

Этот вид машинного обучения долгое время обходили вниманием, пока в Google DeepMind не добились успеха, обучив компьютер играть в игры Atari (а позднее в Go на высочайшем уровне). В обучении с подкреплением *агент* получает информацию о своем окружении и учится выбирать действия, максимизирующие некоторую выгоду. Например, с помощью обучения с подкреплением можно получить нейронную сеть, которая «смотрит» на экран видеоигры и руководит действиями игрока, максимизируя количество очков.

МАШИННОЕ ОБУЧЕНИЕ

ГЛОССАРИЙ КЛАССИФИКАЦИИ И РЕГРЕССИИ

Классификация и регрессия вводят множество специальных терминов. Некоторые из них уже встречались нам в предыдущих примерах, и еще больше их встретится в следующих главах. Они имеют точные определения, характерные для машинного обучения, и вы должны знать их.

- *Образец* (sample), или *вход* (input), — один экземпляр данных, поступающий в модель.
- *Прогноз, предсказание* (prediction), или *выход* (output), — результат работы модели.
- *Цель* (target) — истина. То, что в идеале должна спрогнозировать модель по данным из внешнего источника.
- *Ошибка прогноза* (prediction error), или *значение уровня потерь* (loss value), — мера расстояния между прогнозом модели и целью.
- *Классы* (classes) — набор меток в задаче классификации, доступных для выбора. Например, в задаче классификации изображений с кошками и собаками доступны два класса: «собака» и «кошка».
- *Метка* (label) — конкретный экземпляр класса в задаче классификации. Например, если изображение № 1234 аннотировано как принадлежащее классу «собака», тогда «собака» является меткой для изображения № 1234.
- *Эталоны* (ground-truth), или *аннотации* (annotations), — все цели для набора данных, обычно собранные людьми.
- *Бинарная классификация* (binary classification) — задача классификации, которая должна разделить входные данные на две взаимоисключающие категории.
- *Многоклассовая классификация* (multiclass classification) — задача классификации, которая должна разделить входные данные на более чем две категории. Примером может служить классификация рукописных цифр.
- *Многозначная, или нечеткая, классификация* (multilabel classification) — задача классификации, в которой каждому входному образцу можно присвоить несколько меток. Например, на картинке могут быть изображены кошка и собака вместе, и поэтому такая картинка должна аннотироваться двумя метками: «кошка» и «собака». Количество меток, присваиваемых изображениям, обычно может меняться.
- *Скалярная регрессия* (scalar regression) — задача, в которой цель является скалярным числом, лежащим на непрерывной числовой прямой. Хорошим примером может служить прогнозирование цен на жилье: разные цены из непрерывного диапазона.
- *Векторная регрессия* (vector regression) — задача, в которой цель является набором чисел, лежащих на непрерывной числовой прямой. Например, регрессия по нескольким значениям (таким, как координаты прямоугольника, ограничивающего изображение) является векторной регрессией.
- *Пакет* или *мини-пакет* (batch или mini-batch) — небольшой набор образцов (обычно от 8 до 128), обрабатываемых моделью одновременно. Число образцов часто является степенью двойки для более эффективного использования памяти GPU. В процессе обучения один мини-пакет используется в градиентном спуске для вычисления одного изменения весов модели.

МАШИННОЕ ОБУЧЕНИЕ

Проверка с простым расщеплением выборки

Некоторая часть данных выделяется в контрольный набор. Обучение производится на оставшихся данных, а оценка качества — на контрольных. Как уже говорилось в предыдущих разделах, для предотвращения утечек информации модель не должна настраиваться по результатам прогнозирования на контрольных данных, поэтому требуется *также* зарезервировать отдельный проверочный набор.

Деление данных при использовании проверки с простым расщеплением выборки

Листинг 4.1. Проверка с простым расщеплением выборки

```
num_validation_samples = 10000

np.random.shuffle(data)

validation_data = data[:num_validation_samples]
data = data[num_validation_samples:]

training_data = data[:]

model = get_model()
model.train(training_data)
validation_score = model.evaluate(validation_data)

# В этой точке можно выполнить корректировку модели,
# повторно обучить ее, оценить, повторить корректировку...

model = get_model()
model.train(np.concatenate([training_data,
                            validation_data]))
test_score = model.evaluate(test_data)
```

Перемешивание данных нередко весьма желательно

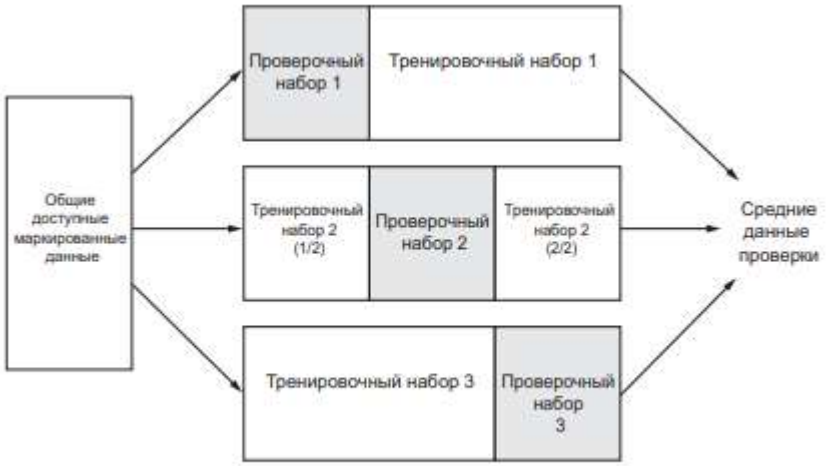
Определение проверочного набора

Определение тренировочного набора

Обучение модели на тренировочных и оценка на проверочных данных

После настройки гиперпараметров часто желательно выполнить обучение окончательной модели на всех данных, не включенных в контрольный набор

Перекрестная проверка по K блокам



Перекрестная проверка по трем блокам

Листинг 4.2. Перекрестная проверка по K блокам

```
k = 4
num_validation_samples = len(data) // k

np.random.shuffle(data)

validation_scores = []
for fold in range(k):
    validation_data = data[num_validation_samples * fold:
      num_validation_samples * (fold + 1)]
    training_data = data[:num_validation_samples * fold] +
      data[num_validation_samples * (fold + 1):]

    model = get_model()
    model.train(training_data)
    validation_score = model.evaluate(validation_data)
    validation_scores.append(validation_score)
    validation_score = np.average(validation_scores)

    model = get_model()
    model.train(data)
    test_score = model.evaluate(test_data)
```

Выбор блока данных для проверки

Использовать остальные данные для обучения. Обратите внимание: оператор + здесь выполняет конкатенацию списков, а не вычисляет сумму

Создание совершенно новой (необученной) модели

Общая оценка: среднее оценок по k блокам

Обучение окончательной модели на всех данных, не вошедших в контрольный набор

МАШИННОЕ ОБУЧЕНИЕ

Что важно помнить

Выбирая протокол оценки, всегда помните:

- *о репрезентативности данных* — наборы тренировочных и контрольных данных должны быть репрезентативными для всего объема имеющихся данных. Например, если вы пытаетесь классифицировать изображения рукописных цифр и имеете массив, в котором образцы упорядочены по классам, использование первых 80 % образцов для обучения и остальных 20 % для контроля приведет к тому, что тренировочный набор будет содержать классы 0–7, а контрольный набор — только классы 8–9. Эта ошибка может показаться смешной, однако ее совершают слишком часто. По этой причине всегда желательно *перемешивать* данные перед делением на тренировочный и контрольный наборы;
- *о направлении оси времени* — пытаясь предсказать будущее по прошлому (например, погоду на завтра, движение товаров и т. д.), вы *не* должны производить перемешивание данных перед делением, потому что это создаст *временну́ю утечку*: ваша модель фактически будет обучаться по данным в будущем. В таких ситуациях всегда нужно следить за тем, чтобы контрольные данные *следовали непосредственно* за тренировочными данными;
- *об избыточности данных* — если некоторые образцы присутствуют в данных в нескольких экземплярах (частое явление в реальном мире), перемешивание и деление данных на тренировочный и проверочный наборы приведет к появлению избыточности между тренировочным и проверочным наборами. По сути, вы будете проводить тестирование на части тренировочных данных, что является худшим из зол! Убедитесь в том, что тренировочный и проверочный наборы не пересекаются.

ОБРАБОТКА ДАННЫХ, КОНСТРУИРОВАНИЕ ПРИЗНАКОВ И ОБУЧЕНИЕ ПРИЗНАКОВ

Векторизация

Все входы и цели в нейронной сети должны быть тензорами чисел с плавающей точкой (или, в особых случаях, тензорами целых чисел). Какие бы данные вам ни требовалось обработать — звук, изображение, текст, — их сначала нужно преобразовать в тензоры. Этот шаг называется *векторизацией данных*. Например, в двух предыдущих примерах классификации текстовых данных мы начали с того, что преобразовали текст в списки целых чисел (представляющие собой последовательности слов) и применили прямое кодирование для превращения списков в тензоры данных типа `float32`. В примерах классификации изображений цифр и предсказания цен на дома исходные данные уже имеют векторизованную форму, поэтому мы пропустили этот шаг.

Нормализация значений

В примере классификации цифр исходные черно-белые изображения цифр были представлены массивами целых чисел в диапазоне 0–255. Прежде чем передать эти данные в сеть, нам понадобилось привести числа к типу `float32` и разделить каждое на 255, в результате чего у нас получились массивы чисел с плавающей точкой в диапазоне 0–1. Аналогично, в примере с предсказанием цен на дома у нас имелись наборы признаков со значениями в разных диапазонах: некоторые признаки были выражены значениями с плавающей точкой, другие — целочисленными значениями. Перед передачей данных в сеть нам понадобилось нормализовать каждый признак в отдельности, чтобы все они имели среднее значение, равное 0, и стандартное отклонение, равное 1.

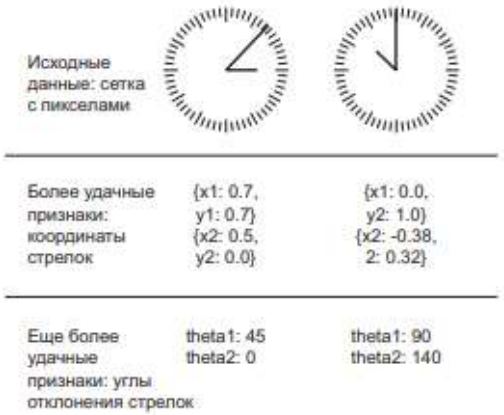
Обработка недостающих значений

Иногда в исходных данных могут отсутствовать некоторые значения. Так, в примере с предсказанием цен на дома первым признаком (столбец с индексом 0 в данных) был уровень преступности на душу населения. Как быть, если этот признак определен не во всех образцах? Если оставить все как есть, у нас будет недоставать значений в тренировочных или в контрольных данных.

Вообще в случае с нейронными сетями вполне безопасно заменить недостающие входные значения нулями, при условии, что 0 не является осмысленным значением. Обрабатывая данные, сеть поймет, что 0 означает отсутствие данных, и будет игнорировать это значение.

Конструирование признаков

Конструирование признаков — это процесс использования ваших собственных знаний о данных и алгоритме машинного обучения (в данном случае — нейронной сети), чтобы улучшить эффективность алгоритма применением предопределенных преобразований к данным перед передачей их в модель. Во многих случаях не следует ожидать, что модель машинного обучения сможет обучиться на полностью произвольных данных. Данные должны передаваться в модель в виде, облегчающем ее работу.



Конструирование признаков для чтения времени с изображения циферблата часов

ПЕРЕОБУЧЕНИЕ И НЕДООБУЧЕНИЕ

Переобучение наблюдается во всех задачах машинного обучения. Умение справляться с этим эффектом играет важную роль в машинном обучении.

Основной проблемой машинного обучения является противоречие между оптимизацией и общностью. Под **оптимизацией** понимается процесс настройки модели для получения максимального качества на тренировочных данных (**обучение в машинном обучении**), а под **общностью** — качество обученной модели на данных, которые она прежде не видела. Цель игры — добиться высокого уровня общности, но вы не можете управлять общностью, вы можете только настраивать модель, опираясь на тренировочные данные.

В начале обучения оптимизация и общность коррелируются: чем ниже потери на тренировочных данных, тем они ниже на контрольных данных. Пока это имеет место, говорят, что модель **недообучена**: прогресс еще возможен, сеть еще не смоделировала все релевантные шаблоны в тренировочных данных. Однако после нескольких итераций на тренировочных данных общность перестает улучшаться, проверочные метрики останавливают свой рост и затем начинают ухудшаться — наступает эффект **переобучения** модели. Другими словами, модель начинает обучаться шаблонам, характерным для тренировочных данных, но нехарактерным для новых данных.

Лучший способ предотвратить изучение моделью специфических или нерелевантных шаблонов, имеющих место в тренировочных данных, — **увеличить объем тренировочных данных**. Модель, обученная на большем объеме данных, будет иметь большую общность. Если это невозможно, следующим лучшим способом является регулирование качества информации или добавление ограничений на информацию, которую модели будет позволено сохранить. Если сеть может позволить себе сохранить только небольшое количество шаблонов, процесс оптимизации заставит ее сосредоточиться на наиболее существенных из них, что увеличит шансы на достижение более высокого уровня общности.

Борьба с переобучением таким способом называется **регуляризацией**.

УМЕНЬШЕНИЕ РАЗМЕРА СЕТИ

В общем случае процесс поиска подходящего размера модели должен начинаться с относительно небольшого количества слоев и параметров, а затем размеры слоев и их количество должны постепенно увеличиваться, пока не произойдет увеличение потерь на проверочных данных.

Давайте опробуем этот подход на сети, выполняющей классификацию отзывов к фильмам. В листинге 4.3 представлена исходная сеть.

Листинг 4.3. Исходная модель

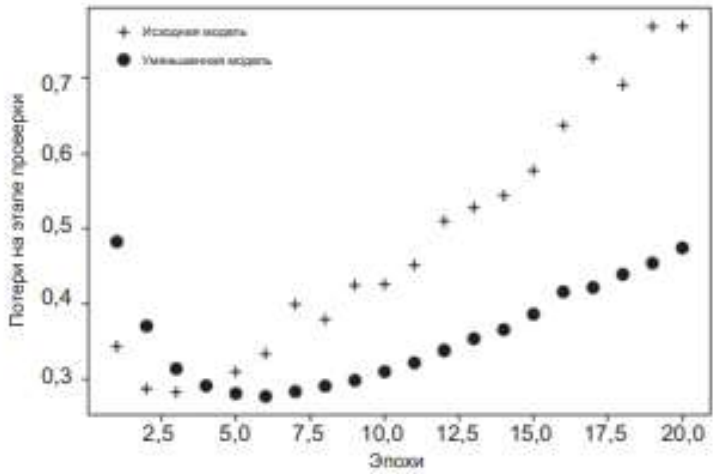
```
from keras import models
from keras import layers

model = models.Sequential()
model.add(layers.Dense(16, activation='relu', input_shape=(10000,)))
model.add(layers.Dense(16, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
```

Теперь попробуем заменить ее меньшей сетью (листинг 4.4).

Листинг 4.4. Версия модели с меньшей емкостью

```
model = models.Sequential()
model.add(layers.Dense(4, activation='relu', input_shape=(10000,)))
model.add(layers.Dense(4, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
```

На рис. представлены для сравнения графики потерь на проверочных данных для исходной сети и уменьшенной ее версии. Точками представлены значения потерь для меньшей сети, а крестиками — для исходной (напомню, что чем ниже потери, тем выше качество модели).



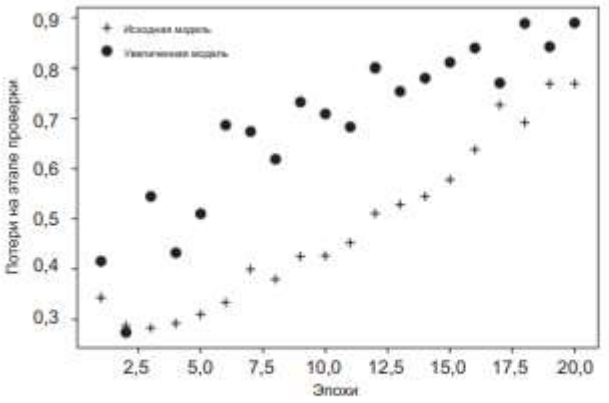
Влияние емкости модели на величину потерь на проверочных данных: попытка уменьшить модель

УМЕНЬШЕНИЕ РАЗМЕРА СЕТИ

Как видите, эффект переобучения уменьшенной сети возникает позже, чем исходной (после шести эпох, а не четырех), и после переобучения ее качество ухудшается более плавно.
Теперь для контраста добавим сеть с большей емкостью — намного большей, чем необходимо для данной задачи (листинг 4.5).

Листинг 4.5. Версия модели с намного большей емкостью

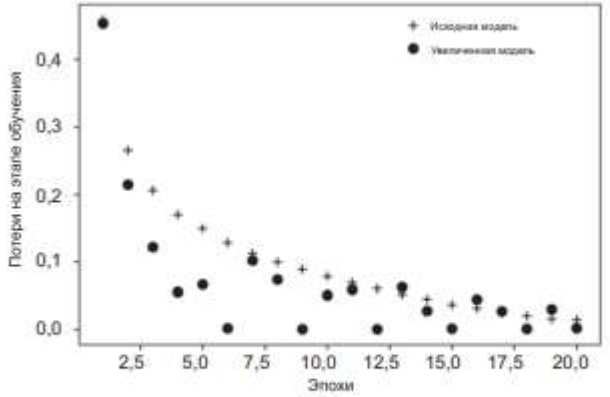
```
model = models.Sequential()
model.add(layers.Dense(512, activation='relu', input_shape=(10000,)))
model.add(layers.Dense(512, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
```



Влияние емкости модели на величину потерь на проверочных данных: попытка увеличить модель

На рис. приводятся для сравнения графики потерь на проверочных данных для большой и исходной сетей. Точками представлены значения потерь для большей сети, а крестиками — для исходной.

Переобучение увеличенной модели наступает почти сразу же, после одной эпохи, и имеет намного более серьезные последствия. Результаты измерения потерь на этапе проверки оказываются слишком искаженными



Влияние емкости модели на величину потерь на тренировочных данных: попытка увеличить модель

ДОБАВЛЕНИЕ РЕГУЛЯРИЗАЦИИ ВЕСОВ

Возможно вы знакомы с принципом *бритвы Оккама*: если какому-то явлению можно дать два объяснения, правильным, скорее всего, будет более простое — имеющее меньшее количество допущений. Эта идея применима также к моделям, полученным с помощью нейронных сетей: для одних и тех же наборов тренировочных данных и архитектуры сети существует множество наборов весовых значений (*моделей*), объясняющих данные. Более простые модели менее склонны к переобучению, чем сложные.

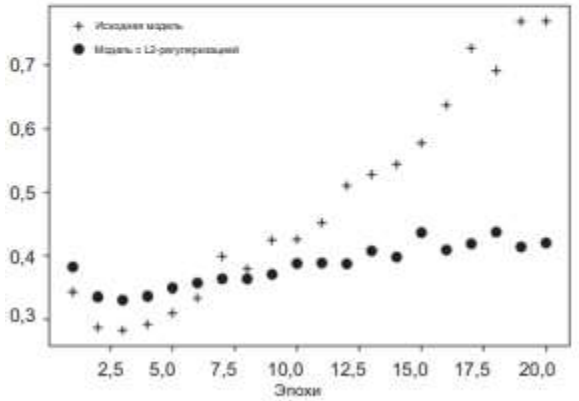
Простая модель в данном контексте — это модель, в которой распределение значений параметров имеет меньшую энтропию (или модель с меньшим числом параметров, как было показано в предыдущем разделе). То есть типичный способ смягчения проблемы переобучения заключается в уменьшении сложности сети путем ограничения значений ее весовых коэффициентов, что делает их распределение более *равномерным*. Этот прием называется *регуляризацией весов*, он реализуется добавлением в функцию потерь сети *штрафа* за увеличение весов и имеет две разновидности:

- *L1-регуляризация* (L1 regularization) — добавляемый штраф прямо пропорционален *абсолютным значениям весовых коэффициентов* (L1-норма весов).
- *L2-регуляризация* (L2 regularization) — добавляемый штраф пропорционален *квадратам значений весовых коэффициентов* (L2-норма весов). В контексте нейронных сетей L2-регуляризация также называется *сокращением весов* (weight decay). Это два разных названия одного и того же явления: сокращение весов с математической точки зрения суть то же самое, что L2-регуляризация.

В Keras регуляризация весов осуществляется путем передачи в слои именованных аргументов с *экземплярами регуляризаторов весов*. Рассмотрим пример добавления L2-регуляризации в сеть классификации отзывов о фильмах (листинг 4.6).

Листинг 4.6. Добавление L2-регуляризации весов в модель

```
from keras import regularizers

model = models.Sequential()
model.add(layers.Dense(16, kernel_regularizer=regularizers.l2(0.001),
                       activation='relu', input_shape=(10000,)))
model.add(layers.Dense(16, kernel_regularizer=regularizers.l2(0.001),
                       activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
```



Влияние L2-регуляризации весов на величину потерь на проверочных данных

Вместо L2-регуляризации можно также использовать следующие регуляризаторы, входящие в состав Keras.

Листинг 4.7. Разные регуляризаторы, доступные в Keras

```
from keras import regularizers

regularizers.l1(0.001)          ← L1-регуляризация

regularizers.l1_l2(l1=0.001, l2=0.001)    ← Объединенная L1- и L2-регуляризация
```

ДОБАВЛЕНИЕ ПРОРЕЖИВАНИЯ

Прореживание (dropout) — один из наиболее эффективных и распространенных приемов регуляризации для нейронных сетей, разработанный Джеффом Хинтоном (Geoff Hinton) и его студентами в Университете Торонто. Прореживание, которое применяется к слою, заключается в *удалении* (присваивании нуля) случайно выбираемым признакам на этапе обучения. Представьте, что в процессе обучения некоторый уровень для данного образца на входе в нормальной ситуации возвращает вектор [0,2, 0,5, 1,3, 0,8, 1,1]. После применения прореживания некоторые элементы вектора получают нулевое значение: например, [0, 0,5, 1,3, 0, 1,1]. *Коэффициент прореживания* — это доля обнуляемых признаков; обычно он выбирается в диапазоне от 0,2 до 0,5. На этапе тестирования прореживание не производится; вместо этого выходные значения уровня уменьшаются на коэффициент, равный коэффициенту прореживания, чтобы компенсировать разницу в активности признаков на этапах тестирования и обучения.

Рассмотрим матрицу Numpy, содержащую результат слоя, `layer_output`, с формой `(размер_пакета, признаки)`. На этапе обучения мы обнуляем случайно выбираемые значения в матрице:

```
layer_output *= np.random.randint(0, high=2, size=layer_output.shape)
```

← На этапе обучения обнуляется 50% признаков в выводе

На этапе тестирования мы уменьшаем результаты на коэффициент прореживания. В данном случае на коэффициент 0,5 (потому что прежде была отброшена половина признаков):

```
layer_output *= 0.5
```

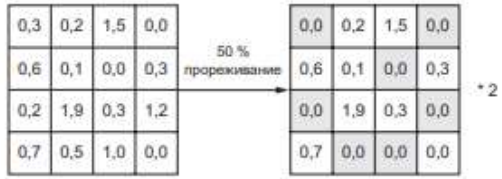
← На этапе тестирования

Обратите внимание на то, что этот процесс можно реализовать полностью на этапе обучения и оставить без изменения результаты, получаемые на этапе тестирования, что часто и делается на практике (рис. 4.8):

```
layer_output *= np.random.randint(0, high=2, size=layer_output.shape)
layer_output /= 0.5
```

← На этапе обучения

← Обратите внимание: в данном случае происходит увеличение, а не уменьшение значений



Прореживание применяется к матрице активации на этапе обучения с масштабированием на этом же этапе. На этапе тестирования матрица активации не изменяется

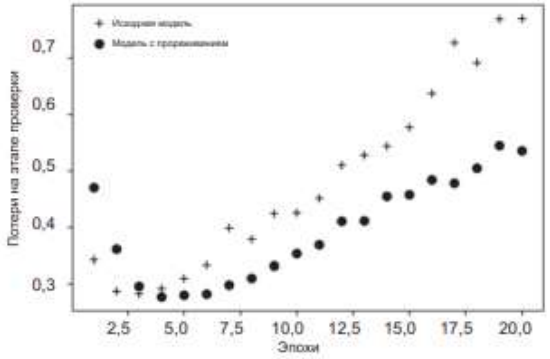
В Keras добавить прореживание в сеть можно посредством уровня `Dropout`, который обрабатывает результаты работы слоя, стоящего непосредственно перед ним:

```
model.add(layers.Dropout(0.5))
```

Давайте добавим два слоя `Dropout` в сеть IMDB и посмотрим, как это повлияет на эффект переобучения.

Листинг 4.8. Добавление прореживания в сеть IMDB

```
model = models.Sequential()
model.add(layers.Dense(16, activation='relu', input_shape=(10000,)))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(16, activation='relu'))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(1, activation='sigmoid'))
```



Влияние прореживания на величину потерь на проверочных данных

ОБОБЩЕННЫЙ ПРОЦЕСС РЕШЕНИЯ ЗАДАЧ МАШИННОГО ОБУЧЕНИЯ

Универсальная схема, которую можно использовать для решения любых задач машинного обучения связывает воедино: определение задачи, оценку, конструирование признаков и ослабление проблемы переобучения

- Определение задачи и создание набора данных
- Выбор меры успеха
- Выбор протокола оценки

После определения цели необходимо также выяснить, как измерять движение к ней. Выше мы рассмотрели три распространенных протокола оценки:

- *выделение из общей выборки отдельного проверочного набора данных* — этот способ хорошо подходит при наличии большого объема данных;
- *перекрестная проверка по K блокам* — правильный выбор при небольшом количестве исходных образцов, из которых нельзя выделить представительную выборку для проверки;
- *итерационная проверка по K блокам с перемешиванием* — позволяет с высокой точностью оценить модель, когда в вашем распоряжении имеется ограниченный объем данных.

Просто выберите один из этих. В большинстве случаев первый поможет получить достаточно надежную оценку.

- Предварительная подготовка данных

После того как определена задача и исходные данные для обучения, цель для оптимизации и порядок оценки предпринятого подхода, у вас есть практически все, чтобы начать обучение модели. Но прежде необходимо преобразовать исходные данные в формат, в котором их можно передать в модель машинного обучения — в данном случае в глубокую нейронную сеть:

- как было показано выше, данные должны быть помещены в тензоры;
- значения, помещаемые в тензоры, обычно требуют масштабирования и приведения к меньшим величинам: например, в диапазоне [–1, 1] или [0, 1];
- если значения разных признаков находятся в разных диапазонах (разнородные данные), их следует нормализовать;
- возможно, вам также понадобится выполнить конструирование признаков, особенно при небольшом объеме исходных данных.

ОБОБЩЕННЫЙ ПРОЦЕСС РЕШЕНИЯ ЗАДАЧ МАШИННОГО ОБУЧЕНИЯ

Ваша цель на этом этапе — достичь статистической мощности, то есть разработать небольшую модель, способную выдать более качественный результат по сравнению с базовым случаем. В примере классификации рукописных цифр из набора MNIST про любую модель, достигшую точности выше 0,1, можно сказать, что она обладает статистической мощностью; в примере IMDB таковой можно назвать любую модель с точностью выше 0,5.

Если все идет как надо, вам нужно сделать три ключевых выбора для создания первой рабочей модели:

- *Функция активации для последнего уровня* — устанавливает эффективные ограничения на результат сети. Например, в случае классификации отзывов из IMDB на последнем уровне используется функция `sigmoid`, в случае регрессии вообще не используется функция активации на последнем уровне и т. д.
- *Функция потерь* — должна соответствовать типу решаемой задачи. Например, в случае IMDB используется функция потерь `binary_crossentropy`, в случае регрессии используется функция `mse` и т. д.
- *Конфигурация оптимизации* — какой оптимизатор использовать? Какой выбрать шаг обучения? В большинстве случаев с успехом можно использовать `rmsprop` с шагом обучения по умолчанию.

Выбор функции активации для последнего уровня и функции потерь

Тип задачи	Функция активации для последнего уровня	Функция потерь
Бинарная классификация	`sigmoid`	`binary_crossentropy`
Многоклассовая, однозначная классификация	`softmax`	`categorical_crossentropy`
Многоклассовая, многозначная классификация	`sigmoid`	`binary_crossentropy`
Регрессия по произвольным значениям	Нет	`mse`
Регрессия по значениям между 0 и 1	`sigmoid`	`mse` или `binary_crossentropy`

Масштабирование по вертикали: разработка модели с переобучением

После получения модели, обладающей статистической мощностью, встает вопрос о достаточной мощности модели.

Чтобы понять, насколько большой должна быть модель, сначала нужно сконструировать модель, обладающую эффектом переобучения. Сделать это просто:

1. Добавьте слои.
2. Задайте большое количество параметров в слоях.
3. Обучите модель на большом количестве эпох.

Постоянно контролируйте, как меняется уровень потерь на этапах обучения и проверки, а также любые другие показатели на этих же этапах, которые вас интересуют. Ухудшение качества модели на проверочных данных свидетельствует о достижении эффекта переобучения.

Регуляризация модели и настройка гиперпараметров

Этот шаг занимает больше всего времени: вам придется многократно изменять свою модель, обучать ее, оценивать качество на проверочных данных (контрольные данные не должны принимать никакого участия в этом этапе), снова изменять ее и повторять этот цикл, пока качество модели не достигнет желаемого уровня. Вот кое-что из того, что вы должны попробовать:

- добавить прореживание;
- опробовать разные архитектуры: добавлять и удалять слои;
- добавить L1- и (или) L2-регуляризацию;
- опробовать разные гиперпараметры (например, число нейронов на слой или шаг обучения оптимизатора), чтобы найти оптимальные настройки;
- дополнительно можно выполнить цикл конструирования признаков: добавить новые признаки или удалить имеющиеся, которые не кажутся информативными.

СПАСИБО ЗА ВНИМАНИЕ!