НЕЙРОСЕТИ НАЧАЛО

НЕЙРОСЕТИ: НОВАЯ ЭРА ОБРАБОТКИ ИНФОРМАЦИИ
УЗНАЙТЕ, КАК НЕЙРОСЕТИ МЕНЯЮТ НАШУ ЖИЗНЬ

ДЖЕЙД КАРТЕР

Джейд Картер
Нейросети начало

http://www.litres.ru/pages/biblio_book/?art=69188950
SelfPub; 2024

Аннотация

Книга является отличным ресурсом для тех, кто хочет познакомиться с основами нейросетей и их применением в жизни. В книге подробно объясняется, что такое нейрон и как он работает в нейросети, что такое веса и смещения, как нейрон принимает решения и как строится нейросеть. Кроме того, книга охватывает такие темы, как обучение нейросетей, основные типы нейросетей (полносвязные, сверточные и рекуррентные), и их применение в задачах классификации, регрессии и кластеризации.Книга также рассматривает продвинутые темы в нейросетях, такие как глубокое обучение, автоэнкодеры и генеративные модели. Автор подробно объясняют, как использовать эти методы в нейросетях и как они могут помочь в решении сложных задач.Независимо от того, являетесь ли вы новичком в области нейросетей или же уже имеете опыт работы с ними, эта книга станет полезным ресурсом для расширения знаний и навыков. Она предоставляет понятную и доступную информацию о технологии, которая становится все более важной в нашей жизни.

Содержание

Джейд Картер
Нейросети начало

Когда машины начинают мыслить, начинается новая глава истории человечества.

Введение

Нейросети – это компьютерные системы, которые пытаются имитировать работу человеческого мозга. Они состоят из нейронов, которые связаны между собой и обрабатывают информацию, передавая ее через нейронные связи. Каждый нейрон выполняет простую функцию, но вместе они могут обрабатывать сложные задачи.

Нейросети важны, потому что они позволяют решать задачи, которые раньше были невозможны или очень трудно решаемы для традиционных методов программирования. Они используются в различных областях, включая обработку изображений и звука, распознавание речи, прогнозирование тенденций в экономике, управление производственными процессами и многое другое.

В настоящее время нейросети являются одним из ключевых элементов машинного обучения и искусственного интеллекта. Они могут обучаться на больших объемах данных

и постепенно улучшать свои результаты, что делает их очень полезными для решения задач, которые ранее были недоступны для автоматизации.

Цель данной книги – познакомить читателя с основами нейросетей, начиная с простых концепций и методов и заканчивая более сложными темами. В книге вы узнаете, как работают нейроны, как обучать нейросети, как выбрать подходящую нейросеть для конкретной задачи, а также применять нейросети для решения задач классификации, регрессии и кластеризации.

Книга рассчитана на начинающих и не требует предварительных знаний в области машинного обучения. Она предоставит читателю полное практическое руководство по работе с нейросетями, которое поможет начать применять их в своих собственных проектах. В процессе чтения книги вы получите необходимые знания и практические навыки для работы с нейросетями, а также узнаете о последних тенденциях и разработках в этой области.

Наша книга поможет вам:

– понять, как работают нейросети и какие задачи они могут решать;

– изучить различные типы нейросетей и выбрать наиболее подходящий для конкретной задачи;

– научиться создавать и обучать нейросети с помощью различных библиотек и инструментов;

– освоить техники работы с данными, подготовки данных и выбора наиболее подходящих параметров модели для достижения наилучших результатов;

– узнать о применении нейросетей в различных областях, таких как обработка изображений, распознавание речи, анализ текста, прогнозирование и многое другое;

– получить практические навыки работы с нейросетями на примерах, которые могут быть применены в реальных проектах.

В этой книге мы сфокусируемся на практическом подходе и предоставим множество примеров и заданий, которые помогут вам лучше понимать и усваивать материал. Вы научитесь создавать нейросети с нуля, обучать их на реальных данных и оценивать их результаты. Мы также предоставим множество ресурсов и ссылок, которые помогут вам продолжить обучение и развиваться в этой области.

Мы уверены, что данная книга будет полезной для всех, кто интересуется нейросетями, машинным обучением и искусственным интеллектом. Независимо от того, являетесь ли вы студентом, профессионалом в области IT или просто любителем технологий, вы найдете в этой книге много полезной информации и практических навыков. Давайте начнем наше путешествие в мир нейросетей!

Глава 1: Основы нейросетей

Нейросети – это мощный инструмент в области искусственного интеллекта и машинного обучения. Они используются во многих приложениях, таких как распознавание речи, обработка изображений и прогнозирование. Однако, чтобы понять, как работает нейросеть, нужно начать с основ.

Основой нейросети является нейрон. Нейрон – это простая единица обработки информации, которая имитирует работу нервной клетки в нашем мозге. Нейрон принимает входные сигналы от других нейронов и генерирует выходной сигнал, который передается другим нейронам.

Каждый нейрон в нейросети имеет веса и смещения. Веса определяют, насколько важен каждый входной сигнал для работы нейрона, а смещение добавляется к сумме входных сигналов, чтобы сделать нейрон более гибким и позволить ему принимать решения в более широком диапазоне входных данных.

Когда нейрон получает входные данные, он умножает их на веса и добавляет смещение. Затем он применяет функцию активации, которая определяет, должен ли нейрон активироваться и передавать сигнал дальше по сети. Функция активации может быть различной в зависимости от задачи, которую решает нейросеть. Например, функция активации может быть сигмоидальной, гиперболического тангенса, ReLU

(Rectified Linear Unit) и многих других.

Нейросеть состоит из множества нейронов, которые объединены в слои. Существует несколько типов слоев, но наиболее распространенные типы слоев – это входной, скрытый и выходной слои. Входной слой принимает входные данные, а выходной слой выдает результат работы нейросети. Скрытые слои находятся между входным и выходным слоями и выполняют различные вычисления, которые помогают нейросети решать задачу.

Когда мы говорим о том, как строится нейросеть, мы имеем в виду, как она объединяет нейроны в слои, как каждый нейрон обрабатывает входные сигналы и какие функции активации используются. Есть множество различных архитектур нейросетей, и выбор конкретной архитектуры зависит от конкретной задачи, которую мы хотим решить.

Важно понимать, что нейросеть обучается путем подстройки весов и смещений для достижения наилучшего результата на тренировочных данных. Обучение нейросети происходит в несколько этапов. На первом этапе мы задаем входные данные и желаемый выходной результат для этих данных. Затем нейросеть прогнозирует результат, и мы сравниваем его с желаемым результатом, чтобы определить ошибку.

С помощью обратного распространения ошибки мы можем корректировать веса и смещения, чтобы уменьшить ошибку и улучшить точность прогнозирования. Этот про-

цесс повторяется множество раз, пока мы не достигнем желаемого уровня точности.

Для более наглядного понимания концепций, которые мы изучили в первой главе, рассмотрим несколько примеров использования нейросетей:

Распознавание цифр на изображениях:

Нейросеть принимает изображение в виде матрицы пикселей размером, скажем, 28x28.

Затем каждый пиксель пропускается через нейрон на первом слое нейросети. Нейрон берет значения пикселей и умножает их на соответствующие веса, затем складывает эти значения и добавляет смещение, и затем применяет функцию активации. Это создает новый набор значений, которые передаются на следующий слой нейросети.

Последующие слои нейросети проходят через этот процесс, используя значения, полученные на предыдущих слоях. Каждый слой может иметь разное количество нейронов и весов, что позволяет нейросети извлекать все более высокоуровневые признаки изображения.

На последнем слое нейросети мы получаем вероятности того, что изображение представляет собой каждую из цифр от 0 до 9. Мы выбираем цифру с наибольшей вероятностью в качестве предсказания нейросети.

Автоматическое распознавание речи:

Нейросеть принимает звуковой файл и разбивает его на последовательности фрагментов. Каждый фрагмент представляет собой короткий участок звука, который может содержать звуковые образцы речи.

Затем каждый фрагмент пропускается через слой нейронов, который использует рекуррентную связь. Это означает, что каждый нейрон хранит в памяти свое предыдущее состояние и использует его для принятия решения на текущем шаге.

После того, как нейросеть обработает все фрагменты звука, мы получим последовательность вероятностей для каждого звукового образца речи в файле. Затем мы используем модель языка, чтобы сгенерировать

Рекомендательная система:

Нейросеть принимает данные о пользователе, такие как их предпочтения, покупки, историю просмотров и т. д.

Затем нейросеть анализирует эти данные и использует их для прогнозирования того, что пользователь может заинтересоваться. Например, если пользователь ранее покупал книги по фантастике, нейросеть может рекомендовать ему

другие книги по этой теме.

Для этого нейросеть может использовать разные типы нейронных сетей, например, сверточные нейронные сети или рекуррентные нейронные сети.

Автоматическое определение эмоций:

Нейросеть принимает данные о голосе, изображении лица или жестах тела человека.

Затем нейросеть анализирует эти данные и использует их для определения эмоционального состояния человека. Например, нейросеть может определить, что человек счастлив, грустен, злится или испытывает другие эмоции.

Для этого нейросеть может использовать сверточные нейронные сети, рекуррентные нейронные сети или комбинацию разных типов сетей.

Это только некоторые примеры того, как нейросети могут быть применены в реальной жизни. Каждый из этих примеров может быть реализован с помощью различных типов нейросетей и конфигураций, и каждый из них может требовать большого объема данных для обучения. Однако, понимание основ работы нейросетей и их структурных элементов, таких как нейроны, веса и функции активации, является ключевым для построения эффективных нейросетей и решения различных задач машинного обучения.

Примеры, описанные в первой главе, могут быть реализованы с помощью различных программных средств для машинного обучения и разработки нейронных сетей. Рассмотрим самые популярные из них.

TensorFlow: это открытое программное обеспечение для машинного обучения, разработанное компанией Google. TensorFlow поддерживает различные типы нейронных сетей и позволяет легко создавать, обучать и развертывать модели машинного обучения.

Keras: это высокоуровневый интерфейс для создания нейронных сетей, который работает поверх TensorFlow. Keras упрощает процесс создания нейросетей и позволяет быстро экспериментировать с разными архитектурами и гиперпараметрами.

PyTorch: это открытое программное обеспечение для машинного обучения, разработанное компанией Facebook. PyTorch также поддерживает различные типы нейронных сетей и обладает удобным интерфейсом для создания и обучения моделей.

Scikit-learn: это библиотека для машинного обучения на языке Python. Scikit-learn включает в себя множество алгоритмов машинного обучения, включая некоторые типы нейронных сетей, и упрощает процесс создания моделей и их оценки.

Конкретный выбор среды для работы зависит от конкрет-

ной задачи и личных предпочтений разработчика. Однако, все эти средства имеют обширную документацию и сообщества пользователей, которые могут помочь в процессе работы с ними.

Рассмотрим более подробно реализацию выше приведенных практических примеров в среде TensorFlow.

Пример кода «Распознавание цифр на изображениях».

Для распознавания цифр на изображениях мы можем использовать нейронную сеть с несколькими сверточными слоями и полносвязными слоями на основе библиотеки TensorFlow. Ниже приведена примерная реализация такой нейронной сети.

Первым шагом является импортирование необходимых модулей TensorFlow и загрузка данных для обучения и тестирования:

```
import tensorflow as tf
from tensorflow import keras
# Загружаем данные MNIST
(train_images, train_labels), (test_images, test_labels) = keras.datasets.mnist.load_data()
```

Затем мы можем преобразовать данные в формат, подходящий для обучения нейронной сети, и нормализовать их.

```
train_images = train_images.reshape((60000, 28, 28, 1))
train_images = train_images / 255.0
test_images = test_images.reshape((10000, 28, 28, 1))
test_images = test_images / 255.0
```

Далее мы можем определить модель нейронной сети. В данном примере мы будем использовать нейронную сеть с тремя сверточными слоями, после каждого из которых применяется слой подвыборки (max pooling), и двумя полносвязными слоями. Выходной слой будет состоять из 10 нейронов, соответствующих классам цифр, и функцией активации softmax.

```
model = keras.Sequential([
keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
keras.layers.MaxPooling2D((2, 2)),
keras.layers.Conv2D(64, (3, 3), activation='relu'),
keras.layers.MaxPooling2D((2, 2)),
keras.layers.Conv2D(64, (3, 3), activation='relu'),
keras.layers.Flatten(),
keras.layers.Dense(64, activation='relu'),
keras.layers.Dense(10, activation='softmax')
```

])

Затем мы можем скомпилировать модель, задав функцию потерь, оптимизатор и метрики для оценки качества модели.

```
model.compile(optimizer='adam',
loss='sparse_categorical_crossentropy',
metrics=['accuracy'])
```

После этого мы можем запустить процесс обучения, передав в модель данные для обучения и тестирования и указав количество эпох (итераций) и размер батча (количество примеров, обрабатываемых за одну итерацию).

```
model.fit(train_images, train_labels, epochs=5, batch_size=64, validation_data=(test_images, test_labels))
```

Наконец, мы можем оценить качество модели на тестовых данных.

```
test_loss, test_acc = model.evaluate(test_images, test_labels)
print('Test accuracy)
```

Результатом обучения нейросети для распознавания цифр на изображениях будет модель, которая способна принимать на вход изображение с рукописной цифрой и предсказывать, какая цифра на изображении изображена.

Этот код позволяет обучить нейросеть для распознавания объектов на изображениях, а именно для классифика-

ции изображений из набора CIFAR-10. Обученная нейросеть может быть использована для распознавания объектов на других изображениях, которые не были использованы в обучающей выборке. Для этого достаточно подать изображение на вход нейросети и получить ответ в виде вероятности принадлежности к каждому из классов.

Для проверки точности модели можно использовать тестовый набор изображений с известными метками (т.е. правильными ответами) и сравнивать предсказания модели с этими метками. Чем выше точность модели на тестовых данных, тем более успешно она справляется с задачей распознавания цифр.

После обучения модели ее можно использовать для распознавания цифр на новых изображениях, например, в приложении для считывания рукописных цифр на почтовых индексах, на банковских чеках или в других сферах, где требуется автоматическое распознавание цифр.

2. Пример кода «Автоматическое распознавание речи».

Для реализации второго примера в среде TensorFlow нам понадобится набор данных CIFAR-10, который можно загрузить с помощью встроенной функции TensorFlow.

Набор CIFAR-10 содержит 60000 цветных изображений размером 32x32 пикселя, разделенных на 10 классов. Для

обучения нейросети мы будем использовать 50000 изображений, а для тестирования – оставшиеся 10000.

Вот как выглядит реализация второго примера в TensorFlow:

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
# Определение архитектуры нейросети
model = keras.Sequential(
[
layers.LSTM(128, input_shape=(None, 13)),
layers.Dense(64, activation="relu"),
layers.Dense(32, activation="relu"),
layers.Dense(10, activation="softmax"),
]
)
# Компиляция модели
model.compile(
optimizer=keras.optimizers.Adam(learning_rate=0.001),
loss=keras.losses.CategoricalCrossentropy(),
metrics=["accuracy"],
)
# Загрузка звукового файла
audio_file = tf.io.read_file("audio.wav")
audio, _ = tf.audio.decode_wav(audio_file)
```

```
    audio = tf.squeeze(audio, axis=-1)
    audio = tf.cast(audio, tf.float32)
    # Разбивка на фрагменты
    frame_length = 640
    frame_step = 320
    audio_length = tf.shape(audio)[0]
    num_frames = tf.cast(tf.math.ceil(audio_length / frame_step), tf.int32)
    padding_length = num_frames * frame_step – audio_length
    audio = tf.pad(audio, [[0, padding_length]])
    audio = tf.reshape(audio, [num_frames, frame_length])
    # Извлечение признаков MFCC
    mfccs = tf.signal.mfccs_from_log_mel_spectrograms(
    tf.math.log(tf.abs(tf.signal.stft(audio))),
    audio.shape[-1],
    num_mel_bins=13,
    dct_coefficient_count=13,
    )
    # Подготовка данных для обучения
    labels = ["one", "two", "three", "four", "five", "six", "seven", "eight", "nine", "zero"]
    label_to_index = dict(zip(labels, range(len(labels))))
    index_to_label = dict(zip(range(len(labels)), labels))
    text = "one two three four five six seven eight nine zero"
    target = tf.keras.preprocessing.text.one_hot(text, len(labels))
    X_train = mfccs[None, …]
```

```
y_train = target[None, …]
# Обучение модели
history = model.fit(X_train, y_train, epochs=10)
# Предсказание результатов
predicted_probs = model.predict(X_train)
predicted_indexes = tf.argmax(predicted_probs, axis=-1)[0]
predicted_labels = [index_to_label[i] for i in
predicted_indexes]
# Вывод результатов
print("Predicted labels:", predicted_labels)
```

Этот код реализует автоматическое распознавание речи с помощью нейросети на основе TensorFlow и Keras. Первым шагом определяется архитектура нейросети с помощью Keras Sequential API. В данном случае используется рекуррентный слой LSTM, принимающий на вход последовательность участков звука длиной 13. Затем идут несколько полносвязных слоев с функцией активации relu и один выходной слой с функцией активации softmax, выдающий вероятности для каждого класса речи.

Далее модель компилируется с помощью метода compile. Оптимизатором выбран Adam с коэффициентом обучения 0.001, функцией потерь – категориальная кросс-энтропия, а в качестве метрики используется точность классификации.

Затем загружается звуковой файл в формате wav, который декодируется с помощью tf.audio.decode_wav и преобразует-

ся в числовые значения float32. Далее происходит разбиение файла на фрагменты длиной 640 с шагом 320. Если файл не делится на равные фрагменты, то добавляется заполнение.

Далее происходит извлечение признаков MFCC (Mel-frequency cepstral coefficients) из каждого фрагмента звука с помощью функции tf.signal.mfccs_from_log_mel_spectrograms. Полученные признаки используются для обучения модели.

Для обучения модели необходимо подготовить данные. В данном случае используется текст с указанием всех возможных классов и соответствующая метка для каждого класса. Для удобства преобразуется текст в one-hot кодировку с помощью метода tf.keras.preprocessing.text.one_hot. Затем подготовленные данные передаются в модель для обучения с помощью метода fit.

После обучения модели предсказываются результаты на тех же данных с помощью метода predict. Выбирается индекс с максимальной вероятностью и соответствующий ему класс.

В конце выводятся предсказанные метки классов.

3. Пример кода «Рекомендательная система».

Для удобства, опишем процесс в пяти шагах:

Шаг 1: Сбор данных

Первый шаг в создании рекомендательной системы – сбор данных. Для этого нужно собрать данные о пользователях,

например, их предпочтения, покупки, историю просмотров и т. д. Эти данные можно получить из различных источников, таких как базы данных или логи пользователей.

Шаг 2: Подготовка данных

После того, как данные собраны, нужно их подготовить. Например, нужно провести предобработку данных, чтобы очистить их от шума и выбросов. Для этого можно использовать различные техники, например, стандартизацию и нормализацию данных.

Шаг 3: Обучение модели

После того, как данные подготовлены, можно перейти к обучению модели. Для создания рекомендательной системы можно использовать различные типы нейронных сетей, например, сверточные нейронные сети или рекуррентные нейронные сети. Модель должна быть обучена на обучающей выборке данных.

Шаг 4: Тестирование модели

После обучения модели, необходимо провести тестирование модели, чтобы убедиться, что она работает правильно. Для этого можно использовать тестовую выборку данных. В процессе тестирования можно провести анализ метрик, таких как точность и полнота.

Шаг 5: Применение модели

После того, как модель прошла тестирование, можно ее применять для рекомендации контента пользователям. Например, можно использовать модель, чтобы рекомендовать

пользователю книги по фантастике, если он ранее покупал такие книги. В этом случае, модель может использовать данные о пользователе, чтобы предсказать, что он может заинтересоваться.

Код решения для рекомендательной системы будет зависеть от того, какие данные о пользователе и предметах рекомендуется использовать, а также какая архитектура нейронной сети будет использоваться. Ниже приведен пример кода для простой рекомендательной системы на основе матричной факторизации, которая использует данные о рейтингах пользователей и предметов:

```
import numpy as np
# загрузка данных
ratings = np.array([
[5, 3, 0, 1],
[4, 0, 0, 1],
[1, 1, 0, 5],
[1, 0, 0, 4],
[0, 1, 5, 4],
])
# инициализация параметров
num_users, num_items = ratings.shape
num_factors = 2
learning_rate = 0.01
num_epochs = 1000
# инициализация матриц пользователей и предметов
```

```
user_matrix = np.random.rand(num_users, num_factors)
item_matrix = np.random.rand(num_factors, num_items)
# обучение матричной факторизации
for epoch in range(num_epochs):
for i in range(num_users):
for j in range(num_items):
if ratings[i][j] > 0:
error = ratings[i][j] – np.dot(user_matrix[i,:], item_matrix[:,j])
user_matrix[i,:] += learning_rate * (error * item_matrix[:,j])
item_matrix[:,j] += learning_rate * (error * user_matrix[i,:])
# прогнозирование рейтингов для всех пользователей и предметов
predicted_ratings = np.dot(user_matrix, item_matrix)
# рекомендация предметов для конкретного пользователя
user_id = 0
recommended_items = np.argsort(predicted_ratings[user_id])[::-1]
print("Рекомендации для пользователя", user_id)
print(recommended_items)
```

В этом примере мы использовали матричную факторизацию для построения рекомендательной системы. Мы инициализировали матрицы пользователей и предметов случайными значениями и обучили их на основе известных рейтингов пользователей и предметов. Затем мы использовали полученные матрицы, чтобы прогнозировать рейтинги для всех

пользователей и предметов, а затем рекомендовали предметы на основе этих прогнозов для конкретного пользователя. В реальных системах могут использоваться более сложные алгоритмы и более разнообразные данные.

4. Пример кода «Автоматическое определение эмоций».

Описание процесса.

Импортируем необходимые модули из TensorFlow.

Создаем модель, используя сверточные нейронные сети. Модель принимает входные данные в виде изображения размером 48x48x1 пикселей. Слои Conv2D, BatchNormalization и MaxPooling2D используются для извлечения признаков из изображения. Слой Flatten преобразует полученные признаки в одномерный вектор. Слои Dense, BatchNormalization и Dropout используются для классификации эмоций на 7 категорий (счастье, грусть, злость и т.д.).

Компилируем модель, указываем оптимизатор, функцию потерь и метрики.

Обучаем модель на обучающем наборе данных с использованием валидационного набора.

Оцениваем точность модели на тестовом наборе данных.

Используем модель для предсказания эмоций на новых данных.

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers

# Создание модели
model = keras.Sequential([
layers.Conv2D(32, (3, 3), activation='relu', input_shape=(48,
48, 1)),
layers.BatchNormalization(),
layers.MaxPooling2D(pool_size=(2, 2)),
layers.Dropout(0.25),
layers.Conv2D(64, (3, 3), activation='relu'),
layers.BatchNormalization(),
layers.MaxPooling2D(pool_size=(2, 2)),
layers.Dropout(0.25),
layers.Conv2D(128, (3, 3), activation='relu'),
layers.BatchNormalization(),
layers.MaxPooling2D(pool_size=(2, 2)),
layers.Dropout(0.25),
layers.Flatten(),
layers.Dense(256, activation='relu'),
layers.BatchNormalization(),
layers.Dropout(0.5),
layers.Dense(7, activation='softmax')
])
# Компиляция модели
```

```
model.compile(optimizer='adam',
loss='categorical_crossentropy', metrics=['accuracy'])
# Обучение модели
history = model.fit(train_data, train_labels, epochs=50,
validation_data=(val_data, val_labels))
# Оценка модели
test_loss, test_acc = model.evaluate(test_data, test_labels)
print('Test accuracy:', test_acc)
# Использование модели
predictions = model.predict(new_data)
```

Этот код создает сверточную нейронную сеть для распознавания эмоций на изображениях размером 48x48 пикселей.

В первом слое используется свертка с 32 фильтрами размера 3x3 и функцией активации ReLU, которая принимает входные изображения размера 48x48x1. Затем следуют слои нормализации пакетов, максимальной пулинги с размером фильтра 2x2 и dropout, который помогает предотвратить переобучение.

Далее добавлены два дополнительных сверточных слоя с увеличенным числом фильтров и аналогичными слоями нормализации и dropout. После этого следует слой сглаживания, который преобразует многомерный вход в одномерный вектор.

Затем следуют два полносвязных слоя с функцией актива-

ции ReLU и функцией нормализации пакетов, а также слои dropout. Последний слой содержит 7 нейронов и использует функцию активации softmax для определения вероятности каждой из 7 эмоций.

Для компиляции модели используется оптимизатор adam, функция потерь categorical_crossentropy и метрика accuracy. Модель обучается на тренировочных данных в течение 50 эпох с валидацией на проверочных данных.

После обучения модели оценивается на тестовых данных и выводится точность предсказаний. Затем используется модель для предсказания эмоций на новых данных.

Итог по 1 главе.

В этой главе мы рассмотрели основные концепции, которые лежат в основе нейросетей. Мы изучили, что такое нейрон, как он работает в нейросети, что такое веса и смещения, как нейрон принимает решения и как строится нейросеть. Мы также рассмотрели процесс обучения нейросети и то, как нейросеть корректирует свои веса и смещения, чтобы улучшить точность прогнозирования.

Итак, можно сделать вывод, что нейросеть – это мощный инструмент в области искусственного интеллекта и машинного обучения, который используется во многих приложениях. Основой нейросети является нейрон, который принимает входные сигналы, обрабатывает их и генерирует выходной

сигнал. Нейросеть состоит из множества нейронов, объединенных в слои, и каждый нейрон имеет веса и смещения.

Мы также рассмотрели практические аспекты создания и обучения нейронных сетей с использованием библиотеки TensorFlow и фреймворка Keras. Мы описали процесс подготовки данных, создания модели, ее компиляции и обучения. Кроме того, мы обсудили важность проверки и оценки модели на тестовых данных.

Коды, которые мы рассмотрели, позволяют создать и обучать нейронную сеть для решения конкретных задач, таких как автоматическое определение эмоций и распознавание изображений, определение эмоций и рекомендательная система. Эти примеры демонстрируют, как можно использовать нейронные сети для решения различных практических задач.

Первая глава предоставляет базовые знания и практические навыки в области нейронных сетей и глубокого обучения, которые могут быть полезны как для начинающих, так и для опытных специалистов в этой области.

Глава 2. Обучение нейросетей

Обучение нейросетей – это процесс настройки параметров нейронной сети на основе примеров входных и выходных данных, чтобы она могла решать задачи, для которых она была создана. Этот процесс включает в себя подгонку параметров модели на основе набора данных, таким образом, чтобы она могла обобщать и делать предсказания для новых данных, которые она ранее не видела.

Зачем нужно обучение нейросетей?

Нейронные сети используются для решения различных задач, таких как классификация изображений, распознавание речи, прогнозирование временных рядов и т.д. Обучение нейросетей является ключевым этапом при создании эффективной и точной модели для решения конкретной задачи. Это позволяет нейросети выявлять сложные зависимости в данных, которые не могут быть легко обнаружены человеком. Кроме того, обучение нейросетей позволяет сократить время, затрачиваемое на ручное создание алгоритмов для решения задач. Таким образом, обучение нейросетей является мощным инструментом для создания автоматизированных систем, способных быстро и точно анализировать данные и принимать решения на основе этого анализа.

Обучение нейросетей может происходить с использовани-

ем различных методов. Каждый метод имеет свои особенности и применяется в зависимости от типа задачи и доступных данных. Рассмотрим некоторые из методов обучения нейросетей:

Backpropagation (обратное распространение ошибки) – один из наиболее распространенных методов обучения нейросетей. Он заключается в передаче данных через нейросеть в прямом направлении (forward pass) для получения выходных значений, а затем в обратном направлении (backward pass) для расчета ошибки и корректировки весов нейронов. Backpropagation позволяет обучать нейросеть на большом количестве данных и дает возможность оптимизировать функцию потерь.

Генетические алгоритмы – это методы обучения, которые используют эволюционный подход. Они применяются для решения задач оптимизации, таких как подбор гиперпараметров или оптимизация функции потерь. Генетические алгоритмы работают с популяцией решений, которая постепенно эволюционирует для достижения лучшего результата.

Стохастический градиентный спуск (Stochastic Gradient Descent, SGD) – это метод, который используется для минимизации функции потерь в нейросети. Он обновляет веса нейронов на каждом шаге, опираясь на градиент функции потерь. SGD особенно полезен при обучении нейросетей на больших наборах данных.

Метод опорных векторов – это метод машинного обучения, который используется для классификации данных. Он заключается в поиске гиперплоскости, которая разделяет данные на два класса. Этот метод может быть полезен для задач классификации, таких как распознавание образов или определение темы текста.

Как выбрать подходящий метод обучения для конкретной задачи? Один из важных факторов – это характеристики данных, с которыми вы работаете. Например, если вы работаете с данными, которые имеют большое количество признаков, то метод опорных векторов может быть полезен для классификации этих данных. Если же вы работаете с данными, которые имеют сложную структуру, например, изображения или звук, то нейронные сети с использованием метода backpropagation могут дать хорошие результаты. Также важно учитывать ограничения по вычислительным ресурсам, доступным для обучения модели.

Функция потерь (англ. loss function) – это математическая функция, которая определяет разницу между выходом нейросети и желаемым выходом. Она используется в процессе обучения нейронной сети для оценки того, насколько точно сеть выполняет задачу, которую требуется ей решить. Функция потерь измеряет ошибку предсказания модели и определяет, какие параметры модели следует настраивать, чтобы

уменьшить эту ошибку.

Различные типы функций потерь используются для разных задач. Например, для задачи классификации обычно используется функция потерь категориальной кросс-энтропии (categorical cross-entropy loss), которая измеряет расстояние между распределением вероятностей, выдаваемым сетью, и правильным распределением классов. Для задачи регрессии, когда требуется предсказать числовое значение, часто используется среднеквадратичная функция потерь (mean squared error loss).

Выбор подходящей функции потерь для конкретной задачи может быть важным шагом в процессе обучения нейросети. Необходимо учитывать тип задачи, количество классов (если это задача классификации), а также особенности данных. Например, если в данных имеются выбросы, то использование среднеквадратичной функции потерь может привести к неустойчивому обучению. В таком случае может быть лучше использовать функцию потерь, которая менее чувствительна к выбросам, например, Huber loss.

Обучение нейросетей происходит в несколько этапов:

Подготовка данных: необходимо подготовить данные для обучения и проверки нейросети. Данные должны быть представлены в виде матриц и векторов, которые можно подать

на вход нейросети. Также необходимо разделить данные на обучающую и проверочную выборки.

Определение архитектуры нейросети: выбор архитектуры нейросети зависит от задачи, которую нужно решить. Необходимо определить количество слоев и количество нейронов в каждом слое, а также выбрать функции активации для каждого слоя.

Инициализация весов: перед началом обучения необходимо инициализировать веса нейросети. Веса могут быть инициализированы случайными значениями или значениями, определенными экспертами.

Обучение нейросети: процесс обучения состоит в постепенной корректировке весов нейросети на основе обучающих данных. Этот процесс осуществляется путем вычисления функции потерь, которая определяет расхождение между предсказанными и фактическими значениями. Для улучшения результатов обучения можно использовать различные методы оптимизации, такие как стохастический градиентный спуск.

Оценка качества модели: после завершения обучения необходимо проверить качество работы модели на проверочной выборке. В этом случае используется метрика, которая определяет, насколько точно модель предсказывает значения.

Финальное тестирование: после того, как модель успешно прошла проверку на проверочной выборке, ее необходимо

протестировать на новых, ранее не виденных данных. Это поможет определить, насколько хорошо модель справляется с задачей.

Оптимизация модели: если результаты не удовлетворительны, можно произвести оптимизацию модели. Это может включать в себя изменение архитектуры нейросети, выбор другой функции потерь, изменение гиперпараметров или использование другого метода оптимизации.

Конечный код, который выполняет все этапы обучения нейросети, может выглядеть примерно так:

```
# Подготовка данных
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# Определение архитектуры нейросети
from keras.models import Sequential
from keras.layers import Dense
model = Sequential()
model.add(Dense(32, activation='relu', input_dim=X_train.shape[1]))
model.add(Dense(16, activation='relu'))
model.add(Dense(1, activation='linear'))
# Инициализация весов
model.compile(loss='mean_squared_error', optimizer='adam')
```

```
# Обучение нейросети
model.fit(X_train, y_train, epochs=100, batch_size=32)
# Оценка качества модели
score = model.evaluate(X_test, y_test)
# Финальное тестирование
y_pred = model.predict(X_new_data)
# Оптимизация модели
model = Sequential()
model.add(Dense(64, activation='relu',
input_dim=X_train.shape[1]))
model.add(Dense(32, activation='relu'))
model.add(Dense(16, activation='relu'))
model.add(Dense(1, activation='linear'))
model.compile(loss='mean_squared_error',
optimizer='adam')
model.fit(X_train, y_train, epochs=200, batch_size=64)
```

Комментарии к коду:

Первая строка импортирует функцию train_test_split из библиотеки Scikit-Learn для разделения данных на обучающую и проверочную выборки.

Далее определяется архитектура нейросети с помощью класса Sequential из библиотеки Keras. Мы создаем модель с двумя скрытыми слоями с 32 и 16 нейронами соответственно, а также одним выходным слоем. Функции активации для скрытых слоев задаются как ReLU, а для выходного слоя

– linear (обычная линейная функция активации). Входной слой определяется автоматически по размерности входных данных.

Затем мы компилируем модель, используя функцию потерь "mean squared error" (среднеквадратичная ошибка) и метод оптимизации "adam".

Далее мы обучаем модель на обучающей выборке, используя метод fit, указывая количество эпох (100) и размер пакета (batch_size=32).

После этого мы оцениваем качество модели на проверочной выборке с помощью метода evaluate и сохраняем результат в переменную score.

Для финального тестирования мы используем обученную модель для предсказания значений на новых данных X_new_data.

Код обучения нейросетей может отличаться в зависимости от используемой библиотеки и языка программирования. Ниже приведен пример кода на языке Python, используя библиотеку TensorFlow:

```
# Импорт библиотек
import tensorflow as tf
import numpy as np
from sklearn.model_selection import train_test_split
# Подготовка данных
```

```
X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
y = np.array([[0], [1], [1], [0]])
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
# Определение архитектуры нейросети
model = tf.keras.Sequential([
tf.keras.layers.Dense(4, input_shape=(2,), activation='relu'),
tf.keras.layers.Dense(1, activation='sigmoid')
])
# Инициализация весов
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
# Обучение нейросети
history = model.fit(X_train, y_train, epochs=1000, validation_data=(X_test, y_test))
# Оценка качества модели
loss, accuracy = model.evaluate(X_test, y_test)
print('Loss:', loss)
print('Accuracy:', accuracy)
# Финальное тестирование
X_new = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
y_pred = model.predict(X_new)
print('Predictions:', y_pred)
```

Конец ознакомительного фрагмента.