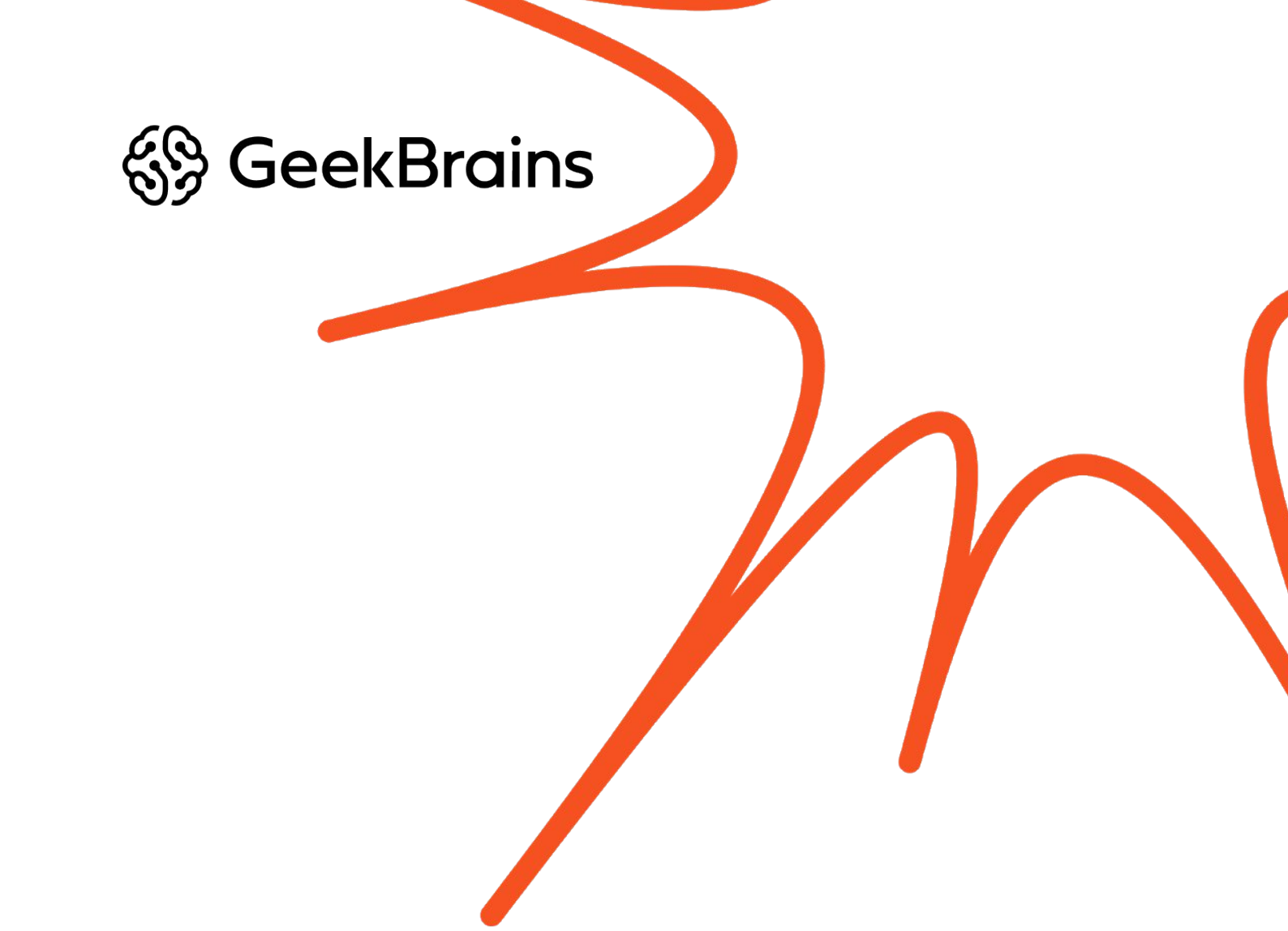

Введение в искусственные нейронные сети

Keras

На этом уроке

1. Изучим способы создания нейросетей
2. Познакомимся с Keras и основами синтаксиса
3. Попрактикуемся создать нейросеть на Keras

Оглавление

[На этом уроке](#)

[Оглавление](#)

[Способы создания нейросетей](#)

[Что такое Keras](#)

[Основы синтаксиса](#)

- [Установка и работа с данными](#)
- [Создание модели](#)
- [Компиляция модели](#)
- [Передача данных для обучения нейросети](#)
- [Оценка обученности нейронной сети](#)
- [Запуск нейронной сети для выполнения работы](#)

[Простая нейросеть на Keras](#)

[Используемые источники](#)

Способы создания нейросетей

Нейросети — это математические модели. Программированием на любом языке можно решать задачи, связанные с математикой. Однако встаёт вопрос, какой язык подойдёт для этого больше. Не считая учебных, нейросети, как правило, работают с большим количеством данных. Чтобы обучение нейросетей происходило с приемлемой скоростью, нужно использовать быстрый язык, например Си. Но такие языки обычно обладают низким уровнем абстракции, программировать и модифицировать на нём нейросети крайне затруднительно.

Для этих целей хорошо может подойти язык Python. С одной стороны, он имеет высокий уровень абстракции, с другой — операции с массивами данных могут сделать его библиотеки, написанные на Си. Этим способом мы пользовались первые 2 урока. Но если таким образом писать нейросети, это приведёт к повторяющемуся коду (поскольку их архитектуры остаются одинаковыми и зачастую меняются только с точки зрения параметров). Для того чтобы реализовывать архитектуры, может понадобиться их хорошее знание архитектур. Такая работа будет затруднительна для людей, не имеющих достаточной подготовки, а для профессионалов — наоборот, будет рутинной.

Существуют фреймворки для создания нейронных сетей (это, пожалуй, основной рабочий способ). Вот их неполный перечень:

- TensorFlow
- PyTorch
- Keras
- Microsoft Cognitive Toolkit (CNTK)
- Caffe
- Apache MXNet

Упрощение создания нейронных сетей на них не заканчивается. Также можно использовать инструменты, позволяющие создавать нейронные сети без навыков программирования, строя нейросети графически. Примеры: Neural Designer, Deep Learning Studio. Кроме того, существуют инструменты, самостоятельно создающие нейронные сети: AutoML инструменты. Самые популярные популярных из них:

- MLBox
- TPOT
- Autokeras

Перечисленные инструменты проранжированы в порядке возрастания уровня абстракции. Говоря о плюсах и минусах того или иного инструмента нужно в первую очередь понимать плюсы и минусы повышения уровня абстракции. Чем он выше, тем меньше его производительность и гибкость, и наоборот.

Наиболее востребованным в рабочих целях является уровень абстракции, который дают фреймворки — мы будем их изучать и использовать. Самый популярный фреймворк для создания нейросетей — TensorFlow. Самый популярный для обучения — Keras. Их мы изучим в этом и последующем уроке.

Также стоит отметить, что фреймворки взаимосвязаны: Keras, как правило, работает поверх TensorFlow, а сам TensorFlow позволяет при необходимости пользоваться средствами Keras.

Что такое Keras

Keras появился относительно недавно — в 2015 г. Но за это время он стал одним из самых популярных фреймоворков для создания нейросетей и фактически стандартом для использования его начинающими.

В чем причина его популярности? Keras позволяет создавать на высоком уровне абстракции — можно оперировать слоями, количеством нейронов в них, выбором функции активации и т.д. В то же время keras содержит инструментарий для всего того, что может понадобиться для работы, например ряд встроенных датасетов и возможность обрабатывать изображения.

В техническом плане Keras — это оболочка над инструментами меньшей степени абстракции, он может работать поверх TensorFlow, Microsoft Cognitive Toolkit, R, Theano, PlaidML.

Keras также пользуется на соревнованиях Kaggle. Однако стоит отметить, что в реальных проектах чаще используется TensorFlow, который мы будем изучать в следующих уроках. Keras, как и любой высокоабстрактный инструмент, имеет изъяны в качестве меньшей гибкости и производительности, чем тот же tensorflow.

Стоит также отметить, что Google официально поддерживает Keras (его автор, François Chollet, является сотрудником Google). TensorFlow сам, в свою очередь, позволяет использовать возможности Keras, т.е. в нём заложена возможность переходить на более высокой уровень абстракции.

В этом уроке мы рассмотрим пример обучения нейронной сети с помощью Keras. Но прежде изучим основы синтаксиса Keras и стандартные задачи, которые нужно выполнить при обучении нейронной сети.

Основы синтаксиса

Установка и работа с данными

Для начала необходимо установить keras. Надо полагать, вы хорошо знакомы с командой pip.

```
sudo python3 pip install keras
```

Давайте попробуем получить датасет mnist и проанализировать его содержимое. Это ещё не будет синтаксис Keras, но мы рассмотрим часто встречающуюся задачу. Не обращайте внимание на предупреждения от TensorFlow: их часто бывает много и при необходимости их можно подавить.

```
import numpy as np
import mnist
import keras

# The first time you run this might be a bit slow, since the
# mnist package has to download and cache the data.
train_images = mnist.train_images()
train_labels = mnist.train_labels()

print(train_images.shape) # (60000, 28, 28)
print(train_labels.shape) # (60000,)
```

Что мы смогли узнать в данном случае? Что тренировочный датасет mnist состоит из 60 000 изображений 28 на 28 пикселей. Такие небольшие датасеты с маленькими изображениями встретятся и в других учебных датасетах.

Что нужно делать теперь? Если скачанный датасет не имеет разделения на тренировочный и тестовый, то поделить их. В нашем случае тренировочный датасет состоит из 60 000 изображений, тестовый — из 10 000, и они поделены по умолчанию.

Теперь нужно конвертировать значения пикселей из вида от 1 до 255 в набор значений от -0.5 до 0.5.

```
import numpy as np
import mnist

train_images = mnist.train_images()
train_labels = mnist.train_labels()
test_images = mnist.test_images()
test_labels = mnist.test_labels()
```

```
# Normalize the images.
train_images = (train_images / 255) - 0.5
test_images = (test_images / 255) - 0.5

# Flatten the images.
train_images = train_images.reshape((-1, 784))
test_images = test_images.reshape((-1, 784))

print(train_images.shape) # (60000, 784)
print(test_images.shape)  # (10000, 784)
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
<ipython-input-1-14f5fe9ee115> in <module>
      1 import numpy as np
----> 2 import mnist
      3
      4 train_images = mnist.train_images()
      5 train_labels = mnist.train_labels()

ModuleNotFoundError: No module named 'mnist'
```

Создание модели

После первичной подготовки данных обычно следует создание модели нейронной сети, которая будет на них учиться.

Ниже типичный код учебной нейросети:

```
# define the keras model
model = Sequential()
model.add(Dense(12, input_dim=8, activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

Разберёмся с командами, которые встретились в этом коде.

- Sequential: позволяет создать нейросети, где слои имеют форму стека. Сигнал в них передаётся от одного слоя к другому. В противовес этой разновидности есть нейросети, где сигнал может передаваться не сразу, а попадать в цикл. Такие нейросети мы разберём в следующих уроках
- Dense: позволяет каждому нейрону быть связанным с другим. В противовес этому может быть необходимость не делать так много связей. Неполносвязные архитектуры также будут разобраны в курсе, они являются основой компьютерного зрения. Цифры 12, 8, 1 обозначают количество нейронов в каждом конкретном слое
- Activation: позволяет определить формулу, по которой будет активироваться нейрон

Компиляция модели

На этапе компиляции создаётся модель с заданными ранее параметрами. Типичный учебный пример:

```
# создание keras модели

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

На этой стадии необходимо сделать дополнительные настройки нейронной сети. Разберём команды из кода выше.

- loss: позволяет задать формулы, по которой будет определяться степень ошибки нейронной сети
- optimizer: позволяет задать алгоритм, который будет осуществлять изменения весов по всей нейронной сети (backpropagation)
- Metrics: позволяет определить критерии, по которым будет оцениваться степень обученности нейросети

Передача данных для обучения нейросети

После того как нейросеть создана, можно передавать ей данные для обучения. Ниже типичный пример кода для этого:

```
# передача обучающего датасета keras модели

model.fit(X, y, epochs=150, batch_size=10, verbose=0)
```

Разберём команды из этого примера. X, y содержат все обучающие данные, epochs определяет, сколько раз весь набор данных должен пройти через нейросеть. bath_size определяет количество обучающих примеров, передающихся нейросети на каждой итерации обучения. verbose позволяет определять информацию, которую вы видите во время обучения нейронной сети.

Оценка обученности нейронной сети

Следующей стадией может быть проверка обученности нейронной сети. Команда Keras для этих целей:

```
results = model.evaluate(x_test, y_test, batch_size=128)
```

В данном случае мы просто указываем, какую модель на каких данных мы хотим проверить.

Запуск нейронной сети для выполнения работы

На этой стадии попробуем запустить нейронную сеть на данных, которые мы бы хотели оценить с её помощью. Например, осуществить распознавание объекта на фотографии. Код для этих целей:

```
predictions = model.predict(x_test[:3])
```

В качестве аргумента здесь указывается массив данных, содержащих, например, фотографию в виде массива чисел.

Мы рассмотрели основные стадии процесса обучения нейросети и команды Keras, в нём задействованные. Конечно, здесь приведён далеко не полный перечень возможностей Keras: она умеет сохранять созданную нейросеть, запускать уже имеющиеся, в ней есть различные средства для создания нейросетей разных архитектур и много другое. С чем-то из арсенала мы разберёмся по ходу курса, а с остальными функциями вы можете познакомиться на сайте Keras в разделе документация.

Простая нейросеть на Keras

Попробуем сделать нейросеть на Keras, используя полученные выше знания. Обучим нейросеть различать рукописные цифры.

```
# The full neural network code!

################################

import numpy as np

import mnist

from keras.models import Sequential
```

```
from keras.layers import Dense
from keras.utils import to_categorical

train_images = mnist.train_images()
train_labels = mnist.train_labels()
test_images = mnist.test_images()
test_labels = mnist.test_labels()

# Normalize the images.
train_images = (train_images / 255) - 0.5
test_images = (test_images / 255) - 0.5

# Flatten the images.
train_images = train_images.reshape((-1, 784))
test_images = test_images.reshape((-1, 784))

# Build the model.
model = Sequential([
  Dense(64, activation='relu', input_shape=(784,)),
  Dense(64, activation='relu'),
  Dense(10, activation='softmax'),
])

# Compile the model.
model.compile(
  optimizer='adam',
  loss='categorical_crossentropy',
  metrics=['accuracy'],
)

# Train the model.
model.fit(
  train_images,
  to_categorical(train_labels),
  epochs=5,
  batch_size=32,
)

# Evaluate the model.
model.evaluate(
```

```
  test_images,
  to_categorical(test_labels)
)

# Save the model to disk.
model.save_weights('model.h5')

# Load the model from disk later using:
# model.load_weights('model.h5')

# Predict on the first 5 test images.
predictions = model.predict(test_images[:5])

# Print our model's predictions.
print(np.argmax(predictions, axis=1)) # [7, 2, 1, 0, 4]

# Check our predictions against the ground truths.
print(test_labels[:5]) # [7, 2, 1, 0, 4]
```

Используемые источники

1. https://keras.io/
2. Шакла Н. — Машинное обучение и TensorFlow 2019