GeekBrains

Введение в искусственные нейронные сети

TensorFlow

На этом уроке

1. Познакомимся с TensorFlow и его основами синтаксиса
2. Рассмотрим его применение на практике

Оглавление

Что такое TensorFlow

TensorFlow — это фреймворк для создания ML моделей. TensorFlow в первую очередь предназначен для Deep Learning, т.е. создания современных нейросетей, однако в нём также есть поддержка некоторых классических ML алгоритмов: K-means clustering, Random Forests, Support Vector Machines, Gaussian Mixture Model clustering, Linear/logistic regression.

TensorFlow выпустила компания Google в 2015 как opensource проект. На данный момент это один из основных инструментов для создания нейросетей в рабочих целях. TensorFlow позволяет создавать нейронные сети как для кластеров из большого количества вычислительных устройств, так и для устройств с относительно небольшой вычислительной мощностью, таких как смартфоны и одноплатные компьютеры.

Google использует TensorFlow для собственных продуктов: поиска, почты, переводчика, распознавания голоса и внутренних нужд, наподобие мониторинга оборудования. Компании применяют TensorFlow для различных проектов, связанных с компьютерным зрением, решением задач ранжирования и т.д.

Основы синтаксиса TensorFlow

Процесс создания нейросети на TensorFlow схож с разобранным нами процессом обучения нейросети на Keras. Отличие заключается в том, что здесь в коде необходимо прописать больше деталей.

Название TensorFlow означает поток тензоров. Тензоры — это массивы. Данные в компьютере часто представлены в виде массивов, и работа с ними подразумевает их преобразование. Преобразования осуществляются через, к примеру, математические операции. Работа TensorFlow складывается из цепочки преобразований тензоров, т.е. данных. Сами операции, осуществляющие преобразование данных, представлены в TensorFlow в виде графов. Особенностью TensorFlow версии 1 является то, что сначала необходимо декларировать переменные и вычисления, которые будут совершены над ними, а уже потом запускать работу над данными.

Рассмотрим базовые вещи в синтаксисе Tensorflow 2. Сначала выведем строку Hello world, а также версию tensorflow:

```
import tensorflow as tf

print(tf.__version__)

msg = tf.constant('TensorFlow 2.0 Hello World')

tf.print(msg)
```

Пример создания тензора:

```
A = tf.constant([[3, 2],
                 [5, 2]])


print(A)
```

Пример сложения тензоров:

```
B = tf.constant([[9, 5],
                 [1, 3]])


AB = tf.concat(values=[A, B], axis=1)
print(AB.numpy())
```

Пример изменения размерности тензора:

```
tensor = tf.constant([[3, 2],
                      [5, 2],
                      [9, 5],
                      [1, 3]])


resh_tensor = tf.reshape(tensor = tensor, shape = [1, 8])


print(f'BEFORE {tensor.numpy()}')
print(f'AFTER {resh_tensor.numpy()}')
```

Пример умножения матриц, одной из самых частых операций в машинном обучении:

```
A = tf.constant([[3, 7],
                 [1, 9]])




B = tf.constant([[10, 10],
                 [1000, 1000]])


AB = tf.multiply(A, B)
print(AB)
```

Классификация изображений одежды

Разберём использование tensorflow 2 на примере датасета с одеждой. В датасете будут находиться маленькие изображения на белом фоне, такие как кроссовки, футболки и прочее. В этом случае мы будем использовать High API от TensorFlow.

```
from __future__ import absolute_import, division, print_function, unicode_literals


# TensorFlow and tf.keras
import tensorflow as tf
from tensorflow import keras

# Helper libraries
import numpy as np
import matplotlib.pyplot as plt

print(tf.__version__)
```

Импортируем Fashion MNIST датасет

Мы будем использовать следующий датасет: Fashion MNIST. Он содержит 70,000 чёрно-белых изображений в 10 категориях, изображения имеют разрешение 28x28 пикселей.

Долгое время в машинном обучении для программ Hello world использовался датасет MNIST с рукописными цифрами. Данный датасет призван несколько усложнить задачу распознавания, но также подходит в качестве программы Hello world.

В этом датасете 60 000 тренировочных изображений и 10 000 тестовых.

```
fashion_mnist = keras.datasets.fashion_mnist


(train_images, train_labels), (test_images, test_labels) = fashion_mnist.load_data()
```

Каждому классу, обозначенному цифрой, мы можем присвоить текстовое значение:

```
class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',

               'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']
```

Анализ датасета

Посмотрим структуры полученного массива данных:

```
train_images.shape

len(train_labels)

train_labels
```

Проанализируем тестовую выборку:

```
test_images.shape

len(test_labels)
```

Preprocess the data

Рассмотрим конкретный пример изображений с помощью `matplotlib`:

```
plt.figure()

plt.imshow(train_images[0])

plt.colorbar()

plt.grid(False)

plt.show()
```

Для процесса обучения нейронной сети важно перевести данные из диапазона от 0 до 255 в диапазон от 0 до 1:

```
train_images = train_images / 255.0



test_images = test_images / 255.0
```

Посмотрим первые 25 изображений:

```
plt.figure(figsize=(10,10))

for i in range(25):

    plt.subplot(5,5,i+1)

    plt.xticks([])

    plt.yticks([])

    plt.grid(False)

    plt.imshow(train_images[i], cmap=plt.cm.binary)

    plt.xlabel(class_names[train_labels[i]])

plt.show()
```

Построение модели

Построение нейронной сети подразумевает конфигурацию её слоев и последующую компиляцию.

Определение слоев

Создадим 3 слоя нейронной сети с помощью функционала `Keras.layers:`

```
model = keras.Sequential([

    keras.layers.Flatten(input_shape=(28, 28)),

    keras.layers.Dense(128, activation='relu'),

    keras.layers.Dense(10)

])
```

Первый слой, `tf.keras.layers.Flatten`, трансформирует двумерный массив на входе в одномерный массив.

Получившиеся 784 (28 x 28) входных нейрона присоединяем к полносвязному слою из 128 нейронов, которые будут использовать функцию активации `relu`. В выходном слое будет 10 нейронов (по числу классов, которые он должен предсказывать). В нём будет использоваться функция активации `softmax`, и он будет давать предсказание от 0 до 1, где 1 — стопроцентная вероятность.

Компиляция модели

Вспомним ключевые понятия, которые понадобятся при компиляции:

- Loss function — измеряет точность работы нейросети
- Optimizer — определяет способ корректировки весов
- Metrics — определяет, какие характеристики будут отражаться в процессе обучения

```
model.compile(optimizer='adam',

              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),

              metrics=['accuracy'])
```

Тренировка модели

Здесь всё стандартно — данные передаются в нейросеть и сопоставляются изображения и лейблы.

Передача данных в модель

Команда, непосредственно запускающая процесс обучения, называется `model.fit`:

```
model.fit(train_images, train_labels, epochs=3)

test_loss, test_acc = model.evaluate(test_images,  test_labels, verbose=2)



print('\nTest accuracy:', test_acc)
```

В выводе выше мы следим за точностью в процессе обучения, проверяем точность на тестовых данных и меняем параметры нейросети, если точность на тестовых данных нас не устраивает.

Предсказания нейросети

Команды ниже позволяют проверить работу натренированной ранее нейросети:

```
probability_model = tf.keras.Sequential([model,

                                         tf.keras.layers.Softmax()])

predictions = probability_model.predict(test_images)

predictions[0]

np.argmax(predictions[0])

test_labels[0]

def plot_image(i, predictions_array, true_label, img):

  predictions_array, true_label, img = predictions_array, true_label[i], img[i]
```

```
  plt.grid(False)
  plt.xticks([])
  plt.yticks([])

  plt.imshow(img, cmap=plt.cm.binary)

  predicted_label = np.argmax(predictions_array)
  if predicted_label == true_label:
    color = 'blue'
  else:
    color = 'red'

  plt.xlabel("{} {:2.0f}% ({})".format(class_names[predicted_label],
                                100*np.max(predictions_array),
                                class_names[true_label]),
                                color=color)

def plot_value_array(i, predictions_array, true_label):
  predictions_array, true_label = predictions_array, true_label[i]
  plt.grid(False)
  plt.xticks(range(10))
  plt.yticks([])
  thisplot = plt.bar(range(10), predictions_array, color="#777777")
  plt.ylim([0, 1])
  predicted_label = np.argmax(predictions_array)

  thisplot[predicted_label].set_color('red')
 thisplot[true_label].set_color('blue')
```

Проверка предсказаний

Matplotlib даёт возможность посмотреть наше предсказание графически:

```
i = 0

plt.figure(figsize=(6,3))

plt.subplot(1,2,1)

plot_image(i, predictions[i], test_labels, test_images)

plt.subplot(1,2,2)

plot_value_array(i, predictions[i],  test_labels)

plt.show()

i = 12

plt.figure(figsize=(6,3))

plt.subplot(1,2,1)

plot_image(i, predictions[i], test_labels, test_images)

plt.subplot(1,2,2)

plot_value_array(i, predictions[i],  test_labels)

plt.show()
```

Сделаем ещё несколько предсказаний:

```
num_rows = 5

num_cols = 3

num_images = num_rows*num_cols

plt.figure(figsize=(2*2*num_cols, 2*num_rows))

for i in range(num_images):

  plt.subplot(num_rows, 2*num_cols, 2*i+1)

  plot_image(i, predictions[i], test_labels, test_images)

  plt.subplot(num_rows, 2*num_cols, 2*i+2)

  plot_value_array(i, predictions[i], test_labels)

plt.tight_layout()

plt.show()
```

Использование полученной модели

Возьмём одно изображение из тестовой выборки и изучим предсказание нейронной сети:

```
img = test_images[1]


print(img.shape)
# Add the image to a batch where it's the only member.
img = (np.expand_dims(img,0))


print(img.shape)
predictions_single = probability_model.predict(img)


print(predictions_single)
plot_value_array(1, predictions_single[0], test_labels)
_ = plt.xticks(range(10), class_names, rotation=45)
```

`keras.Model.predict` возвращает список списков — по одному списку для каждого предсказания в батче. Нам нужны предсказания только для одного изображения:

```
np.argmax(predictions_single[0])
```

При хорошо подобранных параметрах нейросеть должна была выдать корректное предсказание.

Используемые источники

1. https://www.tensorflow.org/
2. https://www.tensorflow.org/tutorials/keras/classification
3. Singh P., Manure A. - LearnTensorFlow 2.0 - 2020
4. Шакла Н. — Машинное обучение и TensorFlow 2019