



Введение в искусственные нейронные сети

TensorFlow

# На этом уроке

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

# Оглавление

[На этом уроке](#_cutwx3tzcepj)

[Оглавление](#_nu4r6zsec6gd)

[Что такое TensorFlow](#_hg9izblzqjkw)

[Основы синтаксиса TensorFlow](#_88o5fe7dq3c3)

[Классификация изображений одежды](#_vy6x4ixjc00d)

[Импортируем Fashion MNIST датасет](#_nbbdzumjsg4j)

[Анализ датасета](#_8o2mjraaqeer)

[Preprocess the data](#_vnmnmhy4f6bg)

[Построение модели](#_e34f3tsp99ts)

[Тренировка модели](#_ek386kq55bb6)

[Использование полученной модели](#_y8xzno1l5we)

[Используемые источники](#_sv0unf2j5zmc)

# 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](https://github.com/zalandoresearch/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