**Модель полносвязной нейроннойсети для задачи многоклассовой классификации**

Рассмотрим создание и обучение модели нейросети, которая классифицирует изображения одежды.

В работе используется библиотека tensorflow.keras - высокоуровневый API для построения и обучения моделей в TensorFlow. Установка библиотек описана в приложении к лабораторной работе 3.

# TensorFlow и tf.keras  
import tensorflow as tf  
from tensorflow import keras  
  
# Вспомогательные библиотеки  
import numpy as np  
import matplotlib.pyplot as plt  
  
print(tf.\_\_version\_\_)

2.0.0

**1. Загрузка набора данных Fashion MNIST**

Набор данных [Fashion MNIST](https://github.com/zalandoresearch/fashion-mnist) содержит 70,000 монохромных изображений в 10 категориях. На каждом изображении содержится по одному предмету одежды в низком разрешении 28 на 28 пикселей (рис.1):



*Рис. 1.* [*Образцы Fashion-MNIST*](https://github.com/zalandoresearch/fashion-mnist) *(Zalando, лицензия MIT).*

Fashion MNIST является альтернативным набором вместо классического набора данных [MNIST](http://yann.lecun.com/exdb/mnist/), который часто используют как "Hello, World" в программах машинного обучения для компьютерного зрения. Набор MNIST содержит изображения рукописных цифр (0, 1, 2, и т.д.). При этом формат изображений одежды идентичен формату изображений цифр.

Набор данных разбит следующим образом:

* 60,000 изображений для обучения нейросети
* 10,000 изображений для проверки модели нейросети, чтобы оценить, насколько правильно сеть обучилась их классифицировать.

Можно получить доступ к Fashion MNIST прямо из TensorFlow, как показано в следующем коде:

fashion\_mnist = keras.datasets.fashion\_mnist

(train\_images, train\_labels), (test\_images, test\_labels) = fashion\_mnist.load\_data()

Процедура загрузки возвращает четыре массива NumPy:

* Массивы train\_images и train\_labels являются *обучающим набором* — данными, на которых модель будет обучаться.
* Массивы *проверочного набора* test\_images и test\_labels.

Изображения (images) являются 28х28 массивами NumPy, где значение пикселей варьируется от 0 до 255. *Метки (labels)*- это массив целых чисел от 0 до 9. Они соответствуют *классам* одежды изображенной на картинках:

|  |  |
| --- | --- |
| Label | Класс |
| 0 | T-shirt/top |
| 1 | Trouser |
| 2 | Pullover |
| 3 | Dress |
| 4 | Coat |
| 5 | Sandal |
| 6 | Shirt |
| 7 | Sneaker |
| 8 | Bag |
| 9 | Ankle boot |

Каждому изображению соответствует единственная метка. Так как *названия классов* не включены в набор данных, заполним массив названиями для дальнейшего использования при построении изображений:

class\_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',  
               'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']

**2. Изучение данных**

Проанализируем формат данных перед обучением модели. Воспользовавшись свойством shape, можно увидеть, что в обучающем наборе данных 60000 изображений, каждое размером 28 x 28 пикселей:

train\_images.shape

(60000, 28, 28)

Соответственно, в обучающем наборе 60000 меток:

len(train\_labels)

60000

Каждая метка это целое число от 0 до 9:

train\_labels

array([9, 0, 0, ..., 3, 0, 5], dtype=uint8)

Проверочный набор содержит 10,000 изображений, каждое имеет такой же размер 28 на 28 пикселей:

test\_images.shape

(10000, 28, 28)

И в проверочном наборе - 10,000 меток:

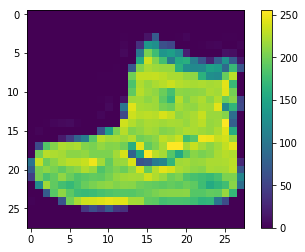
len(test\_labels)

10000

**3. Подготовка данных**

Данные должны быть подготовлены перед обучением нейросети. Если проанализировать первое изображение в обучающем наборе, то можно увидеть, что значения пикселей находятся в диапазоне от 0 до 255. Это видно по изображению первого образца (рис.2)

plt.figure()  
plt.imshow(train\_images[0])  
plt.colorbar()  
plt.grid(False)  
plt.show()



*Рис.2 Отображение одного образца*

Необходимо масштабировать эти значения - привести к диапазону от 0 до 1. Для этого необходимо разделить значения на 255. Важно, чтобы *обучающий набор* и *проверочный набор* были предобработаны одинаково:

train\_images = train\_images/255.0

test\_images = test\_images/255.0

Чтобы убедиться, что данные в правильном формате для построения и обучения нейросети, выведем на экран первые 25 изображений из *обучающего набора* и отобразим под ними наименования их классов (рис.3).

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()



*Рис.3 Изображений из обучающего набора с метками*

**4. Построение модели**

Построение модели нейронной сети требует правильной конфигурации каждого слоя, и последующей компиляции модели.

Базовым строительным блоком нейронной сети является *слой*. Слои извлекают образы из данных, которые в них подаются.

Большая часть глубокого обучения состоит из соединения в последовательность простых слоев. Большинство слоев, таких как tf.keras.layers.Dense, имеют параметры, которые настраиваются во время обучения.

model = keras.Sequential([  
    keras.layers.Flatten(input\_shape=(28, 28)),  
    keras.layers.Dense(128, activation='relu'),  
    keras.layers.Dense(10, activation='softmax')  
])

Первый слой этой сети [tf.keras.layers.Flatten](https://www.tensorflow.org/api_docs/python/tf/keras/layers/Flatten), пробразует формат изображения из двумерного массива (28 на 28 пикселей) в одномерный (размерностью 28 \* 28 = 784 пикселя). Слой извлекает строки пикселей из массива изображения (ранг 2) и выстраивает их в вектор (ранг 1). Этот слой не имеет параметров для обучения; он только переформатирует данные.

После преобразования пикселей, нейросеть содержит два слоя [tf.keras.layers.Dense](https://www.tensorflow.org/api_docs/python/tf/keras/layers/Dense). Это полносвязные нейронные слои. Первый Dense слой состоит из 128 узлов (или нейронов), на выходе слоя используется функция активации *relu*. Второй слой (и последний) имеет 10 узлов, на выходе слоя используется функция активации *softmax*, которая возвращает массив из 10 вероятностных оценок дающих в сумме 1. Т.е. каждый узел выходного слоя содержит оценку указывающую вероятность принадлежности изображения к одному из 10 классов.

**5. Компилирование модели**

Прежде чем модель будет готова для обучения, нужно указать еще несколько параметров. Они добавляются на шаге компилирования содели *compile*:

* *Функция потерь (Loss function)* — измеряет точность модели во время обучения. Требуется минимизировать эту функцию чтобы "направить" модель в верном направлении.
* *Оптимизатор (Optimizer)* — определяет метод обновления модели на основе входных данных и функции потерь.
* *Метрики (Metrics)* — используются для мониторинга обучения и и тестирования модели. Будем использовать метрику *accuracy* равную доле правильно классифицированных изображений.

model.compile(optimizer='adam',  
              loss='sparse\_categorical\_crossentropy',  
              metrics=['accuracy'])

*Примечание.* 1. *adam* - оптимизатор, реализующий алгоритм Адама.

Оптимизация Адама - это метод стохастического градиентного спуска, основанный на адаптивной оценке моментов первого и второго порядка.

Согласно [Kingma et al., 2014](http://arxiv.org/abs/1412.6980), этот метод «эффективен с точки зрения вычислений, требует небольшого объема памяти, инвариантен к диагональному масштабированию градиентов и хорошо подходит для задач, больших с точки зрения данных / параметров »

2. Функции потерь

а) вероятностные функции

*categorical\_crossentropy -* Вычисляет потерю кроссэнтропии между метками и прогнозами. Используйте эту функцию потерь кроссэнтропии, когда существует два или более классов меток. Метки должны быть представлены в формате *one\_hot* (представление категориальных данных в виде двоичных векторов)

*sparse\_categorical\_crossentropy -* вычисляет потерю кроссэнтропии между метками и прогнозами. Используйте эту функцию потерь кроссэнтропии, когда существует два или более классов меток. Метки должны быть представлены как целые числа.

б) регрессионные функции

mean\_squared\_error - Вычисляет средний квадрат ошибок между метками и прогнозами.

3. Кодировка one\_hot

Например имеются 3 категории 'red'-класс 0, 'green' - класс 1, 'blue' - класс 2

Рассмотрим последовательность из 4-х элементов: 'red', 'red', 'green', 'blue'

Представление последовательности в целочисленной кодировке:

0, 0, 1, 2

Представление последовательности в виде one\_hot кодировке (число строк матрицы равно числу элементов последовательности, число колонок равно числу классов, каждая колонка отвечает за свой класс

[1, 0,0

1, 0,0

0, 1,0

0, 0, 1]

**6. Выделение проверочного набора**

Для контроля точности модели создадим проверочный набор, выбрав 10000 образцов из набора обучающих данных (из имеющихся 60000).

x\_val = train\_images[:10000]

partial\_x\_train = train\_images[10000:]

y\_val = train\_labels[:10000]

partial\_y\_train = train\_labels[10000:]

**7. Обучение модели**

Обучение модели нейронной сети требует выполнения следующих шагов:

1. Передать обучающий и проверочные наборы на вход модели. В этом примере обучающие данные: массивы partial\_x\_train и partial\_y\_train, проверочные данные: x\_val и y\_val.
2. Модель учится связывать изображения с правильными классами. Проведем обучение модели в течение 20 эпох.
3. В заключение обученная модель используется для выполнения прогнозов на проверочных данных test\_images. Проверяется, соответствуют ли предсказанные классы меткам из массива test\_labels.

Для начала обучения вызывается метод model.fit, который называется так, поскольку "обучает (fits)" модель на обучающих данных:

history = model.fit(partial\_x\_train,

partial\_y\_train,

epochs=20,

batch\_size=512,

validation\_data=(x\_val, y\_val))

Train on 50000 samples, validate on 10000 samples

Epoch 1/20

50000/50000 [==============================] - 3s 58us/sample - loss: 0.7368 - accuracy: 0.7497 - val\_loss: 0.5142 - val\_accuracy: 0.8268

. . .

Epoch 19/20

50000/50000 [==============================] - ETA: 0s - loss: 0.2578 - accuracy: 0.90 - 1s 27us/sample - loss: 0.2581 - accuracy: 0.9076 - val\_loss: 0.3107 - val\_accuracy: 0.8897

Epoch 20/20

50000/50000 [==============================] - ETA: 0s - loss: 0.2507 - accuracy: 0.9106 E - 1s 28us/sample - loss: 0.2502 - accuracy: 0.9108 - val\_loss: 0.3055 - val\_accuracy: 0.8893

В процессе обучения модели отображаются метрики потери (loss) и точности (accuracy) на обучающих данных, метрики потери (val\_loss) и точности (val\_accuracy) на проверочных данных. Эта модель достигает на обучающих данных точности равной приблизительно 0.91 (91%).

Вызов model.fit() возвращает объект History. Этот объект имеет поле history — словарь с данными обо всем происходившем в процессе обучения. Выведем обозначение основных метрик:

history\_dict = history.history

history\_dict.keys()

dict\_keys(['loss', 'accuracy', 'val\_loss', 'val\_accuracy'])

Словарь содержит четыре элемента— по одному на метрику, — за которыми осуществлялся мониторинг в процессе обучения и проверки. В следующих двух листингах используется библиотека Matplotlib для вывода графиков потерь (рис. 4), а также графиков точности на этапах обучения и проверки (рис. 5).

Результаты могут отличаться от приведенных на рисунках, что обусловлено различием в случайных числах, использовавшихся для инициализации сети.

Выведем графики кривых потерь и точности.

Формирование графиков потерь на этапах обучения и проверки (рис. 4)

loss = history.history['loss']

val\_loss = history.history['val\_loss']

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

plt.plot(epochs, loss, 'bo', label='Потери на этапе обучения')

plt.plot(epochs, val\_loss, 'b', label=' Потери на этапе проверки')

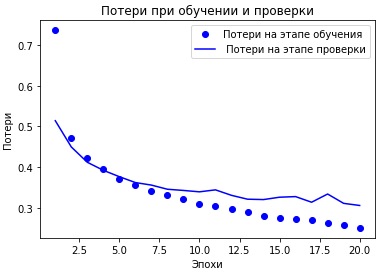
plt.title('Потери при обучении и проверки')

plt.xlabel('Эпохи')

plt.ylabel('Потери')

plt.legend()

plt.show()



*Рис.4. Потери на этапах обучения и проверки*

Формирование графиков точности на этапах обучения и проверки (рис. 5)

plt.clf() **Очистить рисунок**

acc = history.history['accuracy']

val\_acc = history.history['val\_accuracy']

plt.plot(epochs, acc, 'bo', label='Точность на этапе обучения')

plt.plot(epochs, val\_acc, 'b', label=' Точность на этапе проверки')

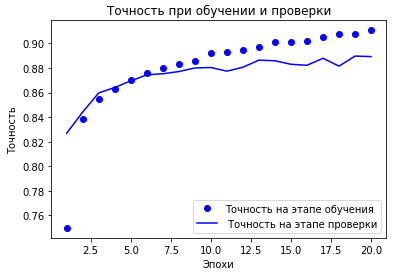
plt.title('Точность при обучении и проверки')

plt.xlabel('Эпохи')

plt.ylabel('Точность')

plt.legend()

plt.show()



*Рис.5. Точность на этапах обучения и проверки*

Но если посмотреть на потери и точность во время проверки модели: после приблизительно 12 эпох они находятся на пике, а далее практически не изменяются и становятся хуже показателей обучения (которые постоянно улучшаются). Это явный пример переобучения: модель показывает более лучшие показатели на данных для обучения, нежели на новых, которых она еще не видела. После этого момента модель начинает переоптимизироваться и обучается представлениям, которые *свойственны* только данным обучения. Таким образом, модель не учится *обобщать* новые, проверочные данные.

**8. Оценка точности модели**

Определим точность модели на контрольном (тестовом) наборе:

test\_loss, test\_acc = model.evaluate(test\_images, test\_labels)  
print('\nТочность на проверочных данных:', test\_acc)

10000/1 [==============================] - 1s 51us/sample - loss: 0.2836 - accuracy: 0.8813

Точность на проверочных данных: 0.8813

Полученная на проверочном наборе точность оказалась немного ниже, чем на обучающем. Этот разрыв между точностью на тренировке и тесте является примером *переобучения (overfitting)*. Переобучение возникает, когда модель машинного обучения показывает на новых данных худший результат, чем на тех, на которых она обучалась.

**9. Использование модели для предсказания**

Теперь, когда модель обучена, ее можно использовать для выполнения предсказаний на контрольном наборе:

predictions = model.predict(test\_images)

В данной команде обученная модель выполняет предсказание класса одежды для каждого изображения в проверочном наборе. Выведем первое предсказание:

predictions[0]

array([2.74253900e-08, 4.62344651e-09, 4.34228042e-10, 6.79940371e-13,

1.26782873e-09, 3.39169892e-05, 1.45382595e-08, 2.83614709e-03,

1.19711075e-07, 9.97129738e-01], dtype=float32)

Прогноз представляет из себя массив из 10 чисел. Они описывают "уверенность" (confidence) модели в том, насколько изображение соответствует каждому из 10 разных видов одежды. Можно посмотреть какой метке соответствует максимальное значение:

np.argmax(predictions[0])

9

Модель полагает, что на первой картинке изображен ботинок (ankle boot), или класс 9. Проверка показывает, что классификация верна:

test\_labels[0]  
9

Построим график, чтобы взглянуть на полный набор из 10 предсказаний классов.

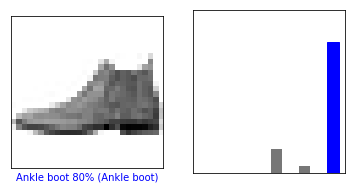
# График изображений

def plot\_image(i, predictions\_array, true\_label, img):  
  predictions\_array, true\_label, img = predictions\_array[i], 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)  
  
# Диаграмма вероятностей классов

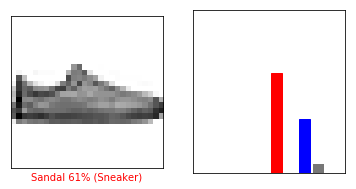
def plot\_value\_array(i, predictions\_array, true\_label):  
  predictions\_array, true\_label = predictions\_array[i], true\_label[i]  
  plt.grid(False)  
  plt.xticks([])  
  plt.yticks([])  
  thisplot = plt.bar(range(10), predictions\_array, color="#777777")  
  plt.ylim([0, 1])  
  predicted\_label = np.argmax(predictions\_array)  
  
  thisplot[predicted\_label].set\_color('red')  
  thisplot[true\_label].set\_color('blue')

Выведем изображение, предсказание и массив предсказаний для образца контрольного набора с индексами 0 и 12.

i = 0  
plt.figure(figsize=(6,3))  
plt.subplot(1,2,1)  
plot\_image(i, predictions, test\_labels, test\_images)  
plt.subplot(1,2,2)  
plot\_value\_array(i, predictions,  test\_labels)  
plt.show()

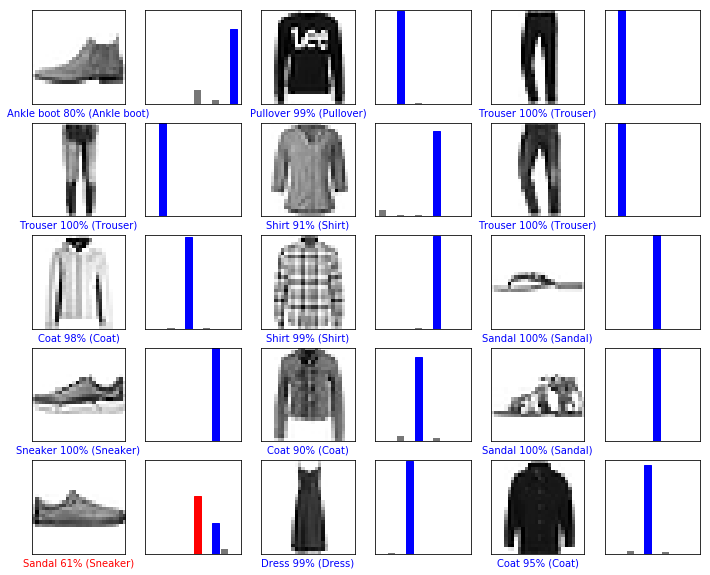


i = 12  
plt.figure(figsize=(6,3))  
plt.subplot(1,2,1)  
plot\_image(i, predictions, test\_labels, test\_images)  
plt.subplot(1,2,2)  
plot\_value\_array(i, predictions,  test\_labels)  
plt.show()



Сверим несколько изображений с их прогнозами. Цвет верных предсказаний синий, а неверных - красный. Число это процент уверенности (от 100) для предсказанной метки. Отметим, что модель может ошибаться, даже если она очень уверена.

# Отображаем первые X тестовых изображений, их предсказанную и настоящую метки.  
# Корректные предсказания окрашиваем в синий цвет, ошибочные в красный.  
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, test\_labels, test\_images)  
  plt.subplot(num\_rows, 2\*num\_cols, 2\*i+2)  
  plot\_value\_array(i, predictions, test\_labels)  
plt.show()



Используем обученную модель для предсказания класса на одном изображении.

# Берем одну картинку из проверочного сета.  
img = test\_images[0]  
print(img.shape)

(28, 28)

Модели tf.keras оптимизированы для предсказаний на *пакетах (batch)* данных, или на множестве примеров сразу. Таким образом, даже если использовать всего 1 картинку, все равно необходимо добавить ее в список:

# Добавляем изображение в пакет данных, состоящий только из одного элемента.  
img = (np.expand\_dims(img,0))  
print(img.shape)

(1, 28, 28)

Предскажем правильную метку для изображения:

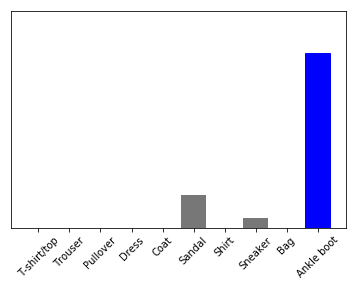
predictions\_single = model.predict(img)  
print(predictions\_single)

[[2.74253900e-08 4.62344651e-09 4.34227210e-10 6.79936467e-13

1.26782873e-09 3.39169565e-05 1.45382595e-08 2.83614174e-03

1.19711075e-07 9.97129738e-01]]

plot\_value\_array(0, predictions\_single, test\_labels)  
\_ = plt.xticks(range(10), class\_names, rotation=45)



Метод model.predict возвращает набор списков, по одному для каждой картинки в пакете данных. Получим прогноз для единственного изображения в пакете:

np.argmax(predictions\_single[0])

9

Т.е. модель предсказывает класс 9.

**Задание.** Постройте модель заданной архитектуры, обучите и проверьте ее точность.

**Дополнительное задание**: Измените параметры архитектуры модели сети и выберите оптимальные значения

* В модели использовались два скрытых слоя. Попробуйте использовать один или три и посмотрите, как это повлияет на ее точность на этапах обучения и проверки.
* Попробуйте использовать первый слой с меньшим количеством скрытых нейронов: 64, 32, 16
* Попробуйте вместо функции потерь *sparse\_categorical\_crossentropy* использовать функцию *binary\_crossentrop*y или *mean\_squared\_error*.
* Попробуйте вместо *relu* использовать функцию активации *tanh* (она была популярна на заре становления нейронных сетей).