

Н. Н. Самылкина

Московский педагогический государственный университет, г. Москва, Россия

И. А. Калинин

Московский государственный лингвистический университет, г. Москва, Россия

ИЗУЧЕНИЕ РАБОТЫ НЕЙРОННОЙ СЕТИ НА ПРИМЕРЕ РАСПОЗНАВАНИЯ РУКОПИСНЫХ ЦИФР С ИСПОЛЬЗОВАНИЕМ ЯЗЫКА ПРОГРАММИРОВАНИЯ PYTHON 3.8 В УГЛУБЛЕННОМ КУРСЕ ИНФОРМАТИКИ

Аннотация

В статье рассматривается возможность включения тем искусственного интеллекта в курс информатики на уровне среднего общего образования на демонстрационном примере распознавания рукописных цифр с использованием языка программирования Python 3.8 и пакета TensorFlow. Пошагово разбирается практический пример задачи классификации с использованием классического набора учебных примеров MNIST (Modified National Institute of Standards and Technology), предложенного Национальным институтом стандартов и технологий США.

Большая часть необходимых для работы средств (интерпретатор языка, основные библиотеки, оболочка для работы) скачивается в виде единого комплекта распространения программного обеспечения (дистрибутива) Anaconda. Тренировка сети происходит разными методами. При обучении модели указываются способ оптимизации, функция потерь и метрика для оценки. Демонстрируется также обработка изображений сверточными сетями, содержащими специальные слои, которые «сворачивают» изображение так, как это делает фильтр растрового изображения.

Ключевые слова: искусственный интеллект, нейронная сеть, сверточная сеть, Python, TensorFlow, преподавание информатики.

DOI: 10.32517/2221-1993-2021-20-10-19-27

1. Введение. Роль и место темы «Искусственный интеллект» в школьном курсе информатики

Анализ развития курса информатики в российской системе общего образования за длительный период времени — более 35 лет — позволил сделать выводы о том,

что темы, относящиеся к искусственному интеллекту, постепенно находят свое место в школьной информатике на уровне среднего общего образования в явном виде. Это означает, что в учебниках не просто есть ознакомительные сведения в виде описаний возможностей экспертных систем (как в недавнем прошлом), а предполагается полноценное изучение тем искусственного

Контактная информация

Самылкина Надежда Николаевна, доктор пед. наук, доцент, профессор кафедры теории и методики обучения математике и информатике, Институт математики и информатики, Московский педагогический государственный университет, г. Москва, Россия; *адрес:* 119991, Россия, г. Москва, ул. Малая Пироговская, д. 1, стр. 1; *e-mail:* nsamylkina@yandex.ru

Калинин Илья Александрович, канд. пед. наук, доцент, начальник управления информатизации, доцент кафедры международной информационной безопасности, Институт информационных наук, Московский государственный лингвистический университет, г. Москва, Россия; *адрес:* 119034, Россия, г. Москва, ул. Остоженка, д. 38, стр. 1; *e-mail:* kalininilya@mail.ru

N. N. Samylkina

Moscow Pedagogical State University, Moscow, Russia

I. A. Kalinin

Moscow State Linguistic University, Moscow, Russia

EXPLORING THE OPERATION OF A NEURAL NETWORK USING THE EXAMPLE OF HANDWRITTEN DIGITS RECOGNITION IN AN ADVANCED INFORMATICS COURSE USING THE PYTHON 3.8 PROGRAMMING LANGUAGE

Abstract

The article discusses the possibility of including artificial intelligence topics in the informatics course at the level of secondary general education on the demonstration example of handwritten digits recognition using the Python 3.8 programming language and the TensorFlow package. A hands-on example of a classification problem using the classic MNIST (Modified National Institute of Standards and Technology) training example set is dealt with step-by-step.

Most of the tools required for the work (language interpreter, basic libraries, and shell) are downloaded in the form of a single software distribution kit of Anaconda. The network is trained using different methods. When the model is trained, the optimization method, the loss function, and the metric for estimation are specified. Image processing by convolutional nets containing special layers, which "convolve" the image the way a bitmap filter does, is also demonstrated.

Keywords: artificial intelligence, neural network, convolutional network, Python, TensorFlow, teaching informatics.

интеллекта с рассмотрением всего спектра решаемых задач этой области, а также выполнением практических работ и учебно-исследовательских проектов [2, 6, 7].

Включению темы «Искусственный интеллект» в курс информатики среднего общего образования способствует несколько причин, среди которых основными являются:

- утверждение государственной программы «Цифровая экономика Российской Федерации» [4] с выделением в ней отдельных сквозных технологий: большие данные, нейротехнологии и искусственный интеллект;
- введение помимо профильной подготовки старшеклассников, где предусмотрена углубленная подготовка по информатике, предпрофессионального обучения старшеклассников в области информационных технологий и инженерной подготовки [8];
- переход на уровне основного общего образования с изучения учебного языка программирования Pascal на многофункциональный язык программирования Python с возможностью продолжения его использования на уровне среднего общего образования [5].

Для будущей успешной профессиональной деятельности в любой сфере сегодняшние выпускники школ должны понимать в контексте развития цифровой экономики, что такое искусственный интеллект, интеллектуальные алгоритмы и анализ больших данных.

Несмотря на имеющиеся публикации о возможностях искусственного интеллекта во многих областях профессиональной деятельности и социальной жизни, а также изучение такой дисциплины в образовательных учреждениях высшего образования, можно констатировать, что пока не существует четко обоснованных методических подходов к изучению вопросов искусственного интеллекта и использованию интеллектуальных алгоритмов анализа данных в *школьном* курсе информатики, в частности, присутствует большой разброс мнений по содержательному наполнению темы, по выбору средств реализации интеллектуальных алгоритмов [9].

2. Нейронные сети в школьном курсе информатики

Одной из интересных и посильных для старшеклассников тем является изучение нейронных сетей в качестве универсального и современного метода в области искусственного интеллекта [3]. Такой курс уже предлагался

в качестве дополнительного к УМК по информатике среднего общего образования [10], но в настоящее время не используется учителями. Именно с нейронными сетями связывают большую часть прогресса в области искусственного интеллекта, который мы на себе ощущаем в последние десять лет, — от голосовых помощников до систем поиска людей по фотографии. Нейронные сети решают множество задач, которые ранее считались крайне трудно разрешимыми, — классификация предметов на изображениях, стилизация текстов, распознавание речи и множество других.

Приведем пример рассмотрения этой темы в рамках классической задачи распознавания рукописных цифр.

Сама идея нейронных сетей достаточно старая, ее разработка была начата учеными еще в прошлом столетии, но только в последние годы стали доступны достаточные ресурсы для реализации действительно эффективных сетей и алгоритмы для их подготовки.

Суть идеи состоит в том, чтобы построить модель биохимического процесса мышления в виде вычислительной модели мозга.

«Аппаратные средства» мозга, как известно из курса биологии, состоят из специальных клеток — нейронов. Нейроны связаны между собой в общую единую сеть своими отростками — дендритами, связанными стыками — аксонами. Вся эта сеть обеспечивает передачу и преобразование электрохимических импульсов, которые и являются физической сутью процесса мышления.

Несмотря на то что представление о реальной физико-химической сущности процесса продолжает изменяться, мы можем для реализации идеи пользоваться таким упрощенным представлением на основе модели Ходжкина-Хаксли. Нейрон получает импульсы, накапливает энергию, после чего ее отдает.

При создании вычислительной нейронной сети нейрон — это логическая единица такой сети. Он имеет несколько входов (чисел), которые суммирует (с учетом «веса» каждого входа), применяет к ним некоторую функцию (чаще всего нелинейную, но простую) и отдает полученное число — результат (рис. 1).

Этот результат поступает на вход другим нейронам и может участвовать в следующем цикле вычислений.

Первая попытка создать такую модель человеческого мозга — это известный в истории вычислительной техники перцептрон.

Нейронные сети, которые мы создаем, как правило, нацелены на решение узкой задачи — например, задачи

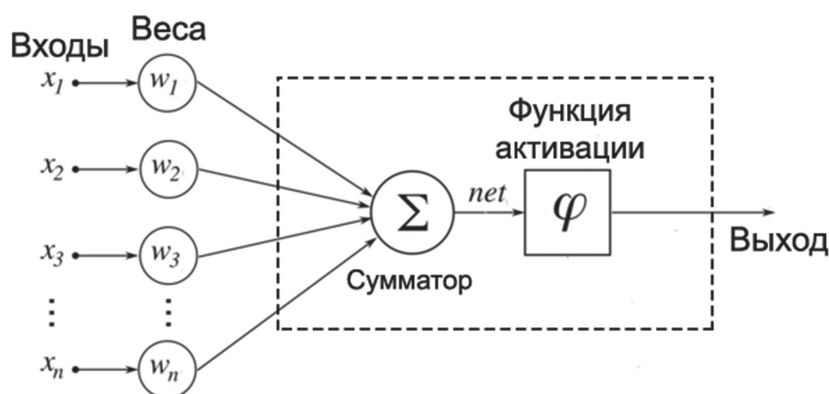


Рис. 1. Схема устройства и работы нейрона вычислительной нейронной сети

распознавания рукописных цифр или классификации каких-либо объектов. Несмотря на узкую постановку задачи, создать такую сеть непросто.

Самая большая трудность состоит в том, что какую бы сеть нейронов мы ни задали изначально, сеть изначально генерирует случайные ответы, поскольку нет способа сразу рассчитать все множество ее параметров. Чтобы добиться от нее приемлемой точности работы, сеть надо — как и другие модели — учить. Суть обучения — изменение весов на входах в нейроны таким образом, чтобы ответы максимально точно совпадали с образцами. Чаще всего это делают с помощью обучения с учителем, т. е. предъявляя готовые примеры с правильными ответами.

Чтобы «учить» сеть, т. е. менять веса, нам необходимо определять до коррекции, насколько сеть ошибается, поэтому для обучения определяется *функция ошибки*. Полученное число будет использоваться, чтобы скорректировать веса. То есть сначала сеть вычислит ответ — это будет называться *прямым распространением*, а потом вычислит ошибку (т. е. оценит, насколько далек вычисленный ответ от настоящего) и выполнит *обратное распространение ошибки*, корректируя вес каждой связи в зависимости от того, как она повлияла на ответ. Поскольку у нас при изменении весов есть цель — минимизировать значение функции ошибки, то мы на самом деле решаем *задачу оптимизации нелинейной функции от огромного множества параметров*. Для решения этой задачи применяют разные вариации *метода градиентного спуска*.

Градиентный спуск, если объяснить очень упрощенно, — это метод, в котором мы постепенно движемся по «поверхности» параметров в ту сторону, в которой целевая функция (ошибка) уменьшается. Шаг такого сдвига мы называем *скоростью обучения*.

Конечно, даже лучшие современные компьютерные нейронные сети и близко не сравнимы по своим возможностям с человеческим мышлением. В первую очередь потому, что требуют десятков и сотен тысяч повторений там, где человеку достаточно максимум десятка примеров. Не говоря уже о том, что человек в состоянии сам вывести закономерности в зависимости от задачи.

3. Задача распознавания рукописных цифр

3.1. Постановка задачи

В качестве тренировочного/демонстрационного примера лучше всего использовать задачу, которая еще

15 лет назад считалась сложной, — распознавание рукописных цифр. Она сводится к *задачам классификации*: имеются различные изображения цифр, а мы должны отнести их к одному из 10 классов. Сейчас эта задача уже считается учебно-демонстрационной, и для нее разработан классический набор учебных примеров, который называется MNIST (Modified National Institute of Standards and Technology), этот набор предложен Национальным институтом стандартов и технологий США (National Institute of Standards and Technology). Это довольно большая база данных образцов рукописного написания цифр — 60 000 изображений размером 28×28 пикселей. Примерный вид образцов представлен на рисунке 2.

3.2. Установка программных средств для работы

Мы будем использовать комплект свободно распространяемого программного обеспечения, состоящий из:

- *интерпретатора языка программирования Python* с некоторым набором расширений — IPython;
- *библиотек алгоритмов* (в том числе интеллектуальных), реализующих всю работу с данными — получение их из разных источников, а также их преобразование и обработку;
- *веб-оболочки*, которая позволит нам создавать документы и с описанием, и с программным кодом, и с результатами его исполнения.

Большую часть необходимых средств (интерпретатор языка, основные библиотеки, оболочку для работы) мы скачаем в виде единого комплекта распространения ПО (дистрибутива) Anaconda. Если не хочется скачивать большой комплект сразу целиком (больше 500 Мбайт), можно скачивать и устанавливать его частями.

Недостающие библиотеки мы установим дополнительно — они также доступны в сети.

1. На сайте: <https://www.anaconda.com/> перейдем в раздел *Download* для скачивания дистрибутива.

2. В разделе *Download* выберем нашу платформу. Будем считать, что у нас основная операционная система — Windows, и скачаем вариант для свежей версии языка Python — Python 3.8.

3. Для управления нашим комплектом и его использования мы можем пользоваться приложением-навигатором (рис. 3).



Рис. 2. Фрагмент образцов базы MNIST

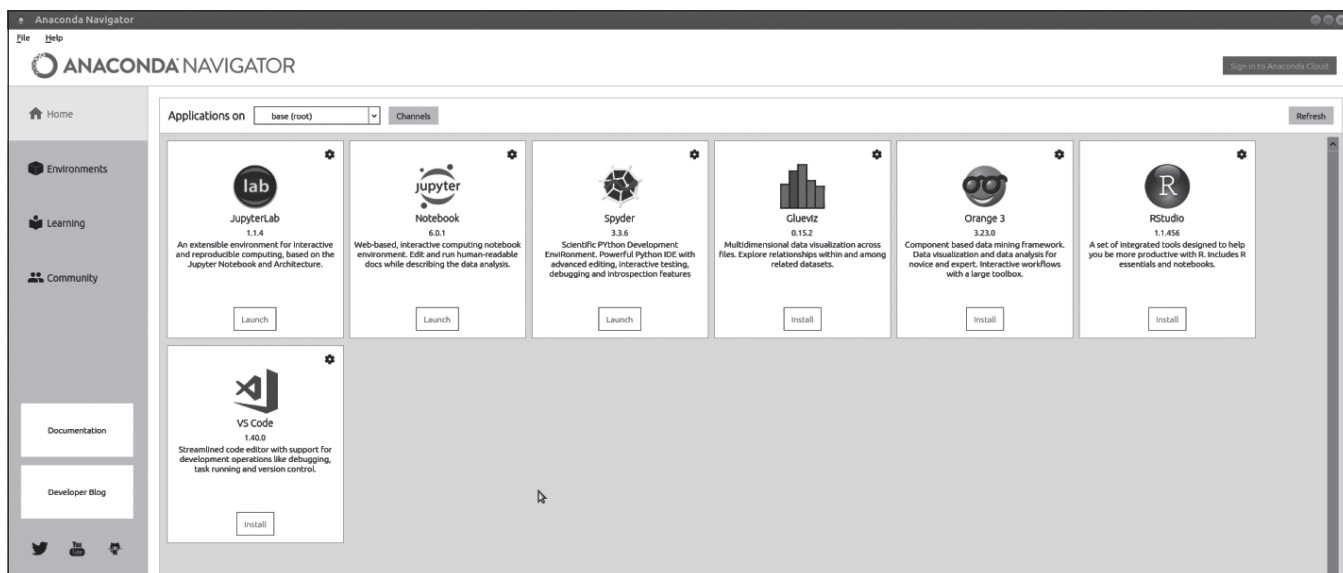


Рис. 3. Приложение-навигатор для запуска Jupyter Notebook

4. Запустив приложение-навигатор, выбираем и запускаем программу Jupyter Notebook (блокнот Jupyter), которая и будет нашим основным рабочим инструментом.

Назначение этой программы — позволить нам описывать рабочий процесс (включая поясняющий текст, формулы, программный код и результаты его исполнения) в виде документа, который называется блокнотом (notebook).

5. После запуска мы видим в рабочей директории страницу со списком файлов и каталогов или пустую (рис. 4).

Чтобы создать свою папку, воспользуемся кнопкой *New* в правом верхнем углу страницы.

6. После нажатия на кнопку *New* нам предложат выбор: создать папку, создать блокнот для работы с интерпретатором Python или создать простой текстовый файл.

Примечание. Блокнот Jupyter состоит из ячеек (cells). Возможны ячейки трех типов:

- 1) *Code* — ячейка с программным кодом, который можно выполнить;
- 2) *Markdown* — ячейка с поясняющим текстом, возможно, с формулами;
- 3) *Raw NBConvert* — ячейка, содержимое которой никак не обрабатывается и не изменяется (например, программный код, который пока не надо исполнять).

При выборе типа ячейки вам покажут четвертый вариант:

- 4) *Heading* — ячейка-заголовок (ячейка типа *Markdown* с символом для формирования заголовка). Это не отдельный тип.

7. Если мы создадим файл блокнота Jupyter с программным кодом на основе языка Python, мы увидим содержимое, как на рисунке 5. То есть это блокнот Jupyter, который называется *Untitled* и состоит из одной ячейки, пока пустой, но потенциально с кодом.

8. Для окончания работы по подготовке среды в нашем демонстрационном блокноте мы выполним команду

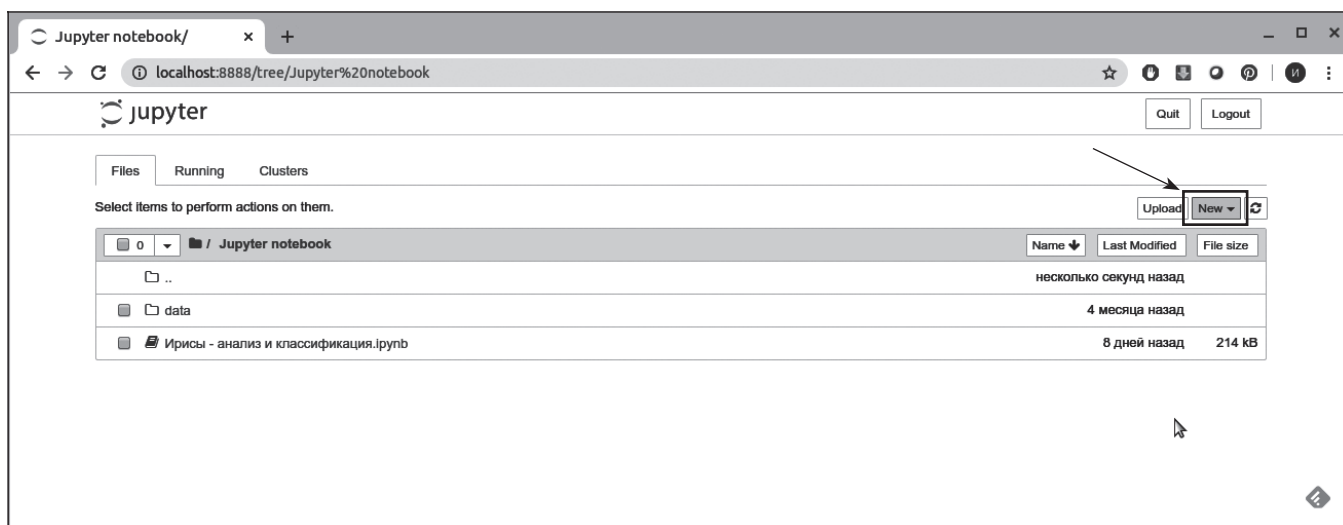


Рис. 4. Список файлов и папок в блокноте Jupyter

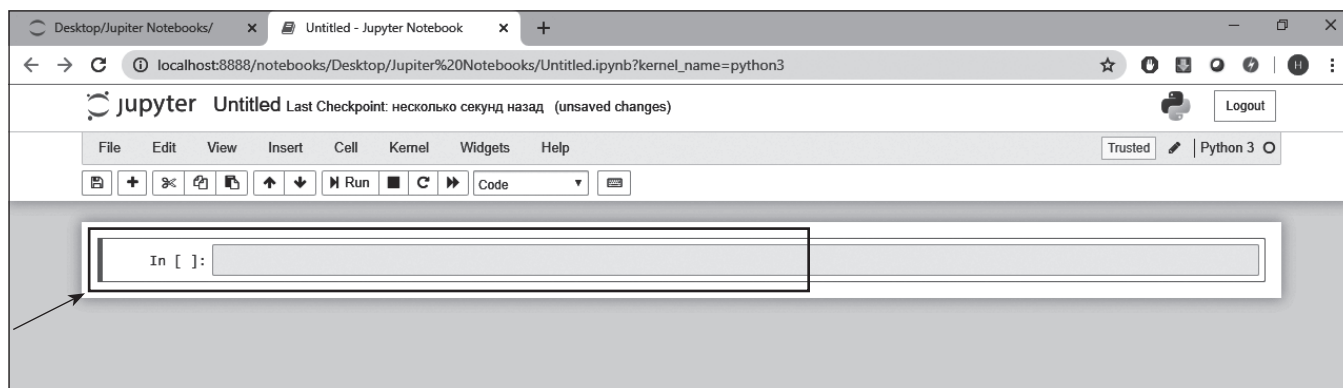


Рис. 5. Пустая ячейка блокнота Jupyter с местом для кода

для установки библиотек. Такую команду мы записываем в ячейку и выполняем. Выполнение, скорее всего, займет некоторое время, и во время выполнения ячейка будет помечена как In [*].

Для установки библиотек ваш компьютер должен иметь доступ к интернету (или вам надо будет заранее подготовить все пакеты и положить их в доступном для среды месте).

9. Для работы с нейронными сетями потребуются библиотеки *TensorFlow* и *Keras*. Поставим их одной командой:

```
!pip install tensorflow
```

10. Библиотека, которую мы будем использовать, содержит уже разработанные функции для загрузки и технической обработки этой базы:

```
# Подключаем необходимые библиотеки
import numpy as np
import os
import tempfile

from tensorflow import keras
from tensorflow.keras import backend as K
from tensorflow.keras import layers
from tensorflow.keras.datasets import mnist

import tensorflow as tf
#tf.enable_eager_execution()

if K.backend() != 'tensorflow':
    raise RuntimeError('Этот пример работает'
        ' только с TensorFlow окружением,'
        ' потому что требует Dataset API,'
        ' которого на других платформах нет.')
```

3.3. Описание нейронной сети

Теперь опишем нашу сеть. Для этого мы будем пользоваться очень известной библиотекой, которую загрузили на предыдущем шаге, — библиотекой *Keras* из пакета *TensorFlow*. При создании сети мы фактически определим поток, т. е. последовательное преобразование.

Начнем с описания общих параметров.

Обучение проводится *эпохами*. Эпоха — это предъявление сети всех примеров из учебного набора, т. е. один цикл обучения. У нас эпох будет пять.

Определяем количество классов на выходе — очевидно, их 10, так как у нас 10 арабских цифр.

Дальше идет *размер мини-пакета*. Мини-пакет — это случайно выбранные примеры из обучающего набора. После того как пакет обработан, библиотека будет проводить нормирование, т. е. накапливать ошибку и предупреждать накопление слишком больших весов. Как ни странно, такое усреднение ошибки заметно улучшает обучаемость сети.

Исходя из этих параметров, определяем количество шагов в эпохе: это общее количество примеров, поделенное на размер мини-пакета.

```
epochs = 5
num_classes = 10

batch_size = 128
buffer_size = 10000
steps_per_epoch = int(np.ceil(60000 /
    float(batch_size))) # = 469

#Загружаем данные из библиотеки. Они уже
#поделены на тестовую и учебную части.
(x_train, y_train), (x_test, y_test) =
    mnist.load_data()

#Все значения цвета в пикселях приводим
#к одному общему типу и шкале - в 255 оттенков.
x_train = x_train / 255.0
x_test = x_test / 255.0
```

Формально сеть может быть соединена связями как угодно. Но на самом деле хаотическая сеть обучению поддаваться будет очень плохо (если вообще будет). Поэтому сети строят из **слоев** — наборов нейронов, которые связаны с нейронами предыдущего и следующего слоев (но не между собой). У любой сети должны быть:

- **входной слой** — нейроны, которые получают исходные данные;
- **выходной слой**, на котором формируется ответ;
- какое-то количество **внутренних слоев**, в которых, собственно, и происходит обработка.

Схематично такую сеть можно изобразить, как представлено на рисунке 6.

На этой схеме показаны не все наши слои, но можно проследить общий принцип: у нас есть:

- входной слой на 784 элемента;
- внутренний скрытый слой (на самом деле, у нас таких слоев четыре);
- выходной слой с функцией *SoftMax* для определения результата.

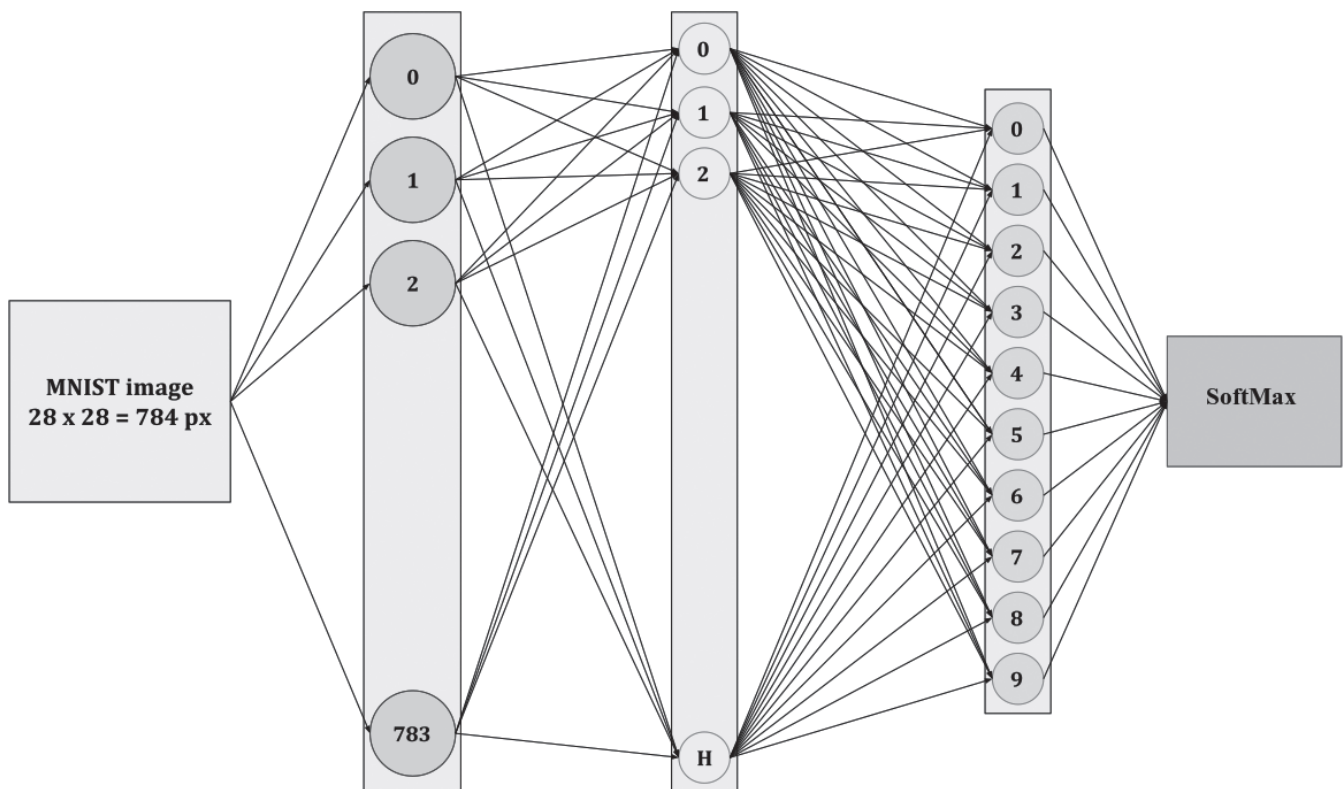


Рис. 6. Схема полносвязной сети для распознавания цифр по образцам MNIST

Есть множество типов слоев, из которых мы используем пока только три:

- *Flatten* — слой, выстраивающий матрицу входных чисел в вектор (входной слой);
- *Dense* — полносвязный слой, в котором на каждый нейрон приходят сигналы со всех нейронов предыдущего слоя;
- *Dropout* — «отбрасывающий» слой, который случайно отбрасывает какую-то долю связей и не пропускает их дальше.

Может показаться странным, но, если у нас не будет отбрасывающего слоя, сеть будет хуже работать с реальными (а не тренировочными) данными. Она будет «переученной», т. е. у нее окажется недостаточно свободы, чтобы правильно формировать ответ на незнакомых данных.

У нас есть несколько возможных функций активации для наших нейронов (рис. 7).

Во внутренних слоях используем самую популярную функцию — полулинейную функцию *ReLU*.

Наша первая сеть устроена очень просто.

Сначала превращаем матрицу изображения в вектор из $28 \times 28 = 784$ чисел (слой типа *Flatten*).

Следующим слоем уменьшаем количество чисел до 512. Это полносвязный слой, т. е. на каждый нейрон придут данные от каждого пикселя. В каждом нейроне мы находим скалярное произведение входного вектора на вектор весов, проводим через функцию активации *ReLU*.

Добавляем еще полносвязные слои, постепенно уменьшая количество нейронов до 128.

Специальным слоем исключаем половину нейронов (т. е. половине нейронов приписываем обязательный 0 для игнорирования значения), чтобы избежать переобучения.

На последнем слое формируем 10 выходных нейронов (поскольку у нас 10 чисел) с функцией *softmax*. Функция *softmax* превратит все веса на последнем слое в числа, сумма которых равна 1, т. е. фактически в вероятности для каждой цифры.

Реализована эта модель так:

```
model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.Dense(256, activation='relu'),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(num_classes,
        activation='softmax')
])
```

1)

Вторую сеть, решающую ту же самую задачу, мы рассмотрим позднее.

3.4. Обучение сети

Теперь мы можем обучать (тренировать) сеть, причем разными методами.

В примере мы можем попробовать разные параметры, а переопределив функцию создания сети, и разные типы сетей.

При обучении модели мы указываем:

- способ оптимизации;
- функцию потерь, т. е. способ определить, насколько фактически вычисленный результат отличается от известного нам. В принципе все обучение нацелено на минимизацию этой функции;
- метрику для оценки. Фактически это параметр, который позволяет сравнить работу двух сетей на нашем наборе данных. Мы используем метрику «Точности», т. е. долю верных ответов.

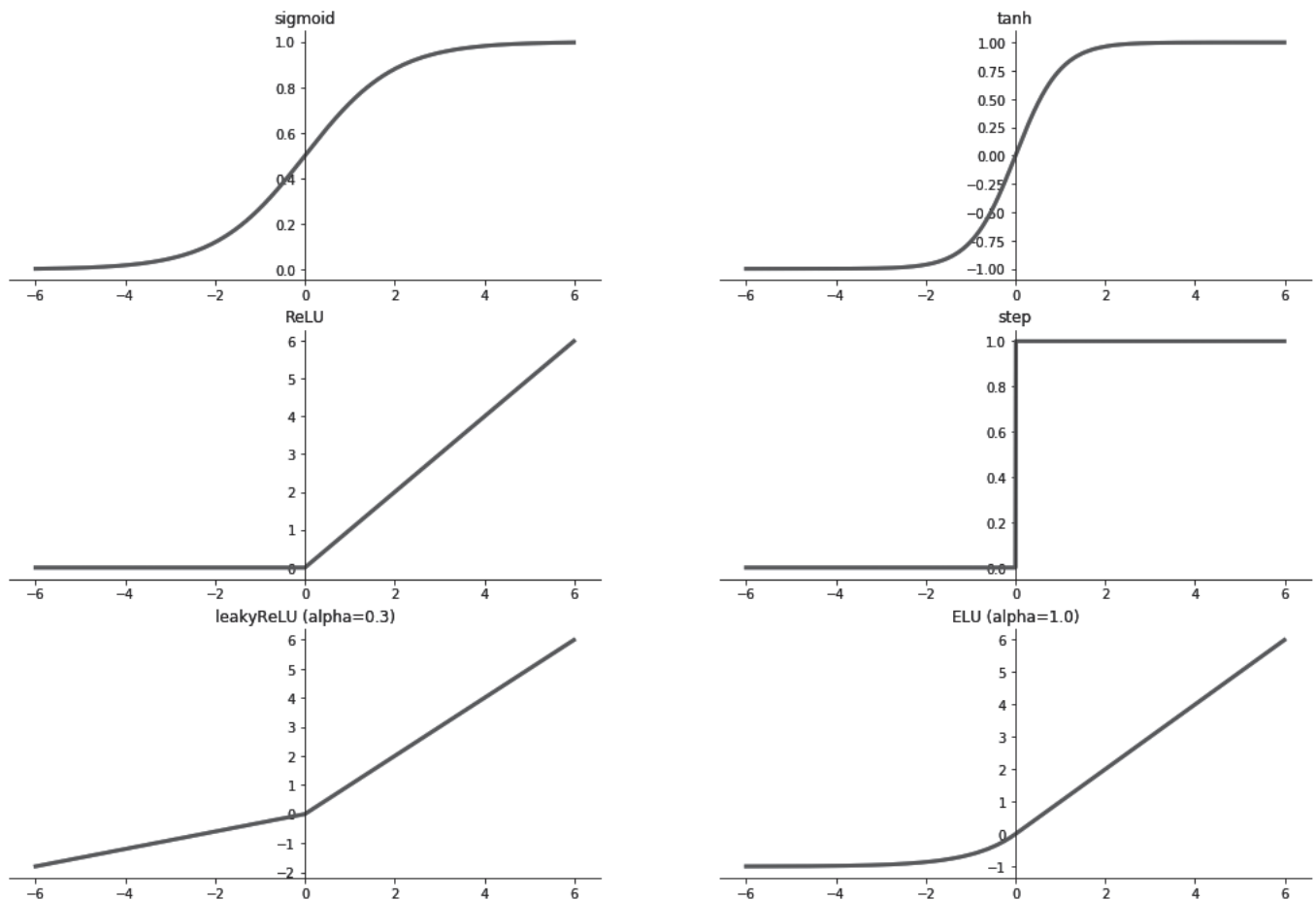


Рис. 7. Функции активации и их графики

Ниже представлена программа обучения сети:

Компилируем модель, т. е. готовим ее к работе.

```
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
# Выведем структуру нашей сети
model.summary()

# Для сверточной сети:
x_train = tf.reshape(x_train, shape=[-1, 28, 28, 1])
# Тренируем сеть
model.fit(x_train, y_train, epochs=epochs,
         batch_size=batch_size,
         steps_per_epoch=steps_per_epoch)
```

Результат работы программы выглядит, как показано на рисунке 8.

Во-первых, здесь мы видим, сколько параметров прошло оптимизацию — 567 434. Во-вторых, можно было видеть, как растет точность с каждой эпохой — на тренировочном наборе мы добрались до точности в 98.69 %.

Теперь посмотрим, что произойдет на тестовом наборе:

```
# Сохраняем сеть, т. е. все полученные веса
weight_path = os.path.join(tempfile.gettempdir(),
                             'saved_wt.h5')
model.save_weights(weight_path)
```

```
loss, acc = model.evaluate(x_test, y_test,
                           num_classes=2)
print('\nTest accuracy: {0}'.format(acc))
```

У авторов получилось округленно 97.82 %. Скорее всего, у читателей получится близкое значение. Поскольку обучение сети — процесс отчасти случайный, этот показатель может быть чуть лучше или хуже, но порядок будет сохраняться.

4. Сверточные сети

Второй тип сетей, с которым мы познакомимся, — **сверточные сети** [1]. В первую очередь они предназначены для обработки изображений, т. е. это как раз наш случай. Такая сеть содержит специальные слои, которые «сворачивают» изображение так, как это делает фильтр растрового изображения. Каждый такой **сверточный слой** выделяет из изображения признаки, например границы объектов (рис. 9).

Наш первый слой:

```
Conv2D(32, (3, 3), activation='relu',
      padding='valid')(input_shape=(28, 28))
```

Мы создадим слой из 32 фильтров размером 3×3 пикселя. Каждый из фильтров будет применяться ко всему изображению и работать так: он накладывается на изображение, все значения цветов умножаются на веса из матрицы фильтра и суммируются. Как и в любой

Model: "sequential_1"

Layer (type)	Output Shape	Param #
flatten_1 (Flatten)	(None, 784)	0
dense_4 (Dense)	(None, 512)	401920
dense_5 (Dense)	(None, 256)	131328
dense_6 (Dense)	(None, 128)	32896
dropout_1 (Dropout)	(None, 128)	0
dense_7 (Dense)	(None, 10)	1290

Total params: 567,434

Trainable params: 567,434

Non-trainable params: 0

Train on 60000 samples

Epoch 1/5

60000/60000 [=====] - 7s 109us/sample - loss: 0.3049 - accuracy: 0.9097

Epoch 2/5

60000/60000 [=====] - 6s 92us/sample - loss: 0.1103 - accuracy: 0.9679

Epoch 3/5

60000/60000 [=====] - 5s 90us/sample - loss: 0.0738 - accuracy: 0.9786

Epoch 4/5

60000/60000 [=====] - 5s 85us/sample - loss: 0.0543 - accuracy: 0.9837

Epoch 5/5

60000/60000 [=====] - 6s 92us/sample - loss: 0.0447 - accuracy: 0.9869

Рис. 8. Результат работы программы

нейронной сети, добавляем число-смещение, и эта сумма поступает на вход функции активации *ReLU*. Так что мы создаем 32 маленьких слоя, т. е. будем обучать $32 \times (3 \times 3 + 1) = 320$ параметров. Важный вопрос: а как мы будем накладывать наш фильтр на края, ведь 28 на-

цело не делится на 3? Мы выбираем вариант наложения «только настоящие значения» — *valid*, со сдвигом на 1 (граница фильтра совпадает с границей изображения). Так что на выходе получится изображение размером 26×26 точек.

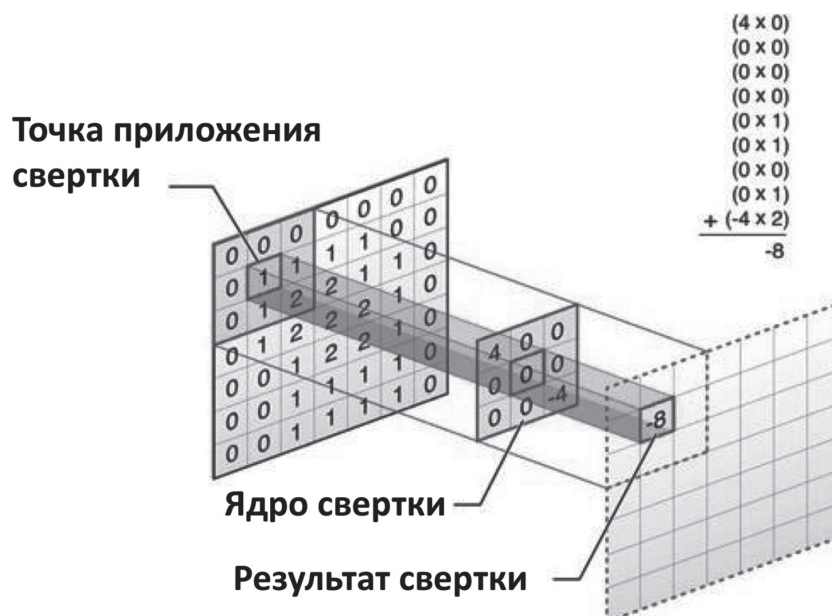


Рис. 9. Принцип действия фильтра-свертки для сверточной нейронной сети

После каждой такой обработки полученное отфильтрованное изображение мы еще уменьшим, создав слой:

```
MaxPooling2D(pool_size=(2, 2))
```

Этот слой параметров не имеет, он накладывает на изображение сетку 2×2 и выбирает максимальное значение из этих четырех пикселей. Изображение уменьшается вдвое до 13×13 пикселей.

Повторим эту пару операций, создав слой, который будет искать уже 64 признака из какой-то комбинации предыдущих:

```
Conv2D(64, (3, 3), activation='relu')
MaxPooling2D(pool_size=(2, 2))
```

И снова уменьшим количество параметров на выходе.

Теперь, выделив признаки второго порядка, развернем их в обычный вектор и отправим на уже знакомый полносвязный слой, а потом отбросим половину и выберем максимальный признак. Реализована такая модель следующим программным кодом:

```
# Сверточная сеть
model = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(32, (3, 3),
        activation='relu', padding='valid',
        input_shape=(28, 28, 1)),
    tf.keras.layers.MaxPooling2D(pool_size=(2, 2)),
    tf.keras.layers.Conv2D(64, (3, 3),
        activation='relu'),
    tf.keras.layers.MaxPooling2D(pool_size=(2, 2)),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(num_classes,
        activation='softmax')
])
```

Процедура обучения сети для нас не изменится (но есть некоторые изменения в подготовке данных — после загрузки данных нужно добавить строку `x_train = x_train.reshape(-1, 28, 28, 1)`), так что мы перделаем описание модели и выучим новую сеть (т. е. перезапустим все блоки кода, начиная с третьего), а потом сравним точность.

Следует учесть, что библиотеки обновляются, поэтому, возможно, на вашем компьютере придется анализировать ошибки и вносить небольшие изменения в программный код.

Прогон показывает, что точность такой сети оказывается значительно выше — на тестовой выборке она оказывается близка к 99 % (в разных прогонах у авторов получалось на тестовой выборке от 98,8 % до 99,12 %), т. е. количество ошибок уменьшается в два-три раза. Лучшие образцы таких сетей показывают результаты до 99,7 % точности, так что есть куда стремиться.

Особенно важно то, что такие сети позволяют обрабатывать большие изображения, поскольку параметров в них гораздо меньше. Именно они используются для

распознавания лиц, предметов и других аналогичных задач.

5. Вопросы и задания для самостоятельной работы учащихся

1. Как повлияет на точность увеличение или уменьшение количества эпох?
2. Как повлияет на точность изменение размеров мини-батча?
3. Попробуйте поменять функции активации и методы оптимизации. Что больше повлияло на скорость и точность?
4. Воспользуйтесь другим тестовым набором — Fashion MNIST и обучите сеть распознавать предметы одежды.

6. Заключение

Как можно видеть, приведенный пример рассмотрения темы, с одной стороны, не требует от учащихся каких-либо компетенций, выходящих за рамки углубленного школьного курса информатики, а с другой стороны, демонстрирует решение практической задачи, предоставляет возможности для экспериментов и проектной деятельности, углубления в современную отраслевую тематику, что, с точки зрения авторов, создает устойчивую базу для дальнейшего профессионального самоопределения учащихся.

Список использованных источников

1. Базовая классификация: классифицируйте изображения одежды. <https://www.tensorflow.org/tutorials/keras/classification>
2. Калинин И. А., Самылкина Н. Н. Интеллектуальные алгоритмы и искусственный интеллект // Информатика. 2014. № 10. С. 38–47.
3. Калинин И. А., Самылкина Н. Н. Информатика. 11 класс. Углубленный уровень: учебник. 2-е изд., стереотип. М.: БИНОМ. Лаборатория знаний, 2020. 212 с.
4. Национальная программа «Цифровая экономика Российской Федерации». <http://static.government.ru/media/files/9gFM4FhJ4PsB79I5v7yLVuPgu4bvR7M0.pdf>
5. Примерная основная образовательная программа основного общего образования. Одобрена решением федерального учебно-методического объединения по общему образованию (протокол от 8 апреля 2015 г. № 1/15). <http://fgosreestr.ru/wp-content/uploads/2017/03/primernaja-osnovnaja-obrazovatel'naja-programma-osnovnogo-obshchego-obrazovanija.pdf>
6. Самылкина Н. Н. Организация углубленного обучения информатике на основе интегративного подхода: монография. М.: МПГУ, 2020. 346 с.
7. Самылкина Н. Н., Салахова А. А. Основы искусственного интеллекта в школьном курсе информатики: история вопроса и направления развития // Информатика в школе. 2019. № 7. С. 32–39. DOI: 10.32517/2221-1993-2019-18-7-32-39
8. Федеральный государственный образовательный стандарт среднего общего образования. <https://fgos.ru/>
9. Ясницкий Л. Н. Введение в искусственный интеллект: учебное пособие для студентов высших учебных заведений. 2-е изд., испр. М.: Академия, 2008. 176 с.
10. Ясницкий Л. Н. Искусственный интеллект. Элективный курс: учебное пособие. М.: БИНОМ. Лаборатория знаний, 2012. 197 с.