МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования
«Юго-Западный государственный университет»
(ЮЗГУ)

Кафедра вычислительной техники

УТВЕРЖДАЮ
Проректор по учебной работе
_____ О.Г. Локтионова
«15» 01 2021 г.

ПРИМЕНЕНИЕ ИСКУССТВЕННЫХ НЕЙРОННЫХ СЕТЕЙ ГЛУБОКОГО ОБУЧЕНИЯ ДЛЯ КЛАССИФИКАЦИИ ИЗОБРАЖЕНИЙ ЦИФР

Методические рекомендации по выполнению лабораторных работ для студентов направления 09.04.01

Курск 2021

УДК 004.652

Составители: И.Е. Чернецкая, С.Ю. Мирошниченко

Рецензент

Кандидат технических наук, доцент *Ю.А. Халин*

Применение искусственных нейронных сетей глубокого обучения для классификации изображений цифр: методические рекомендации по выполнению лабораторных работ / Юго-Зап. гос. ун-т; сост.: С.Ю. Мирошниченко, И.Е. Чернецкая. – Курск, 2021. – 22 с. – Библиогр.: с. 22.

Содержит сведения по вопросам развертывания и применения искусственных нейронных сетей глубокого обучения для классификации изображений рукописных цифр. Указывается порядок выполнения цикла лабораторных работ, рассматриваются вопросы настройки программной среды для работы с нейронными сетями, описываются процессы обучения и распознавания изображений.

Методические рекомендации соответствуют рабочей программе дисциплины «Современные проблемы информатики и вычислительной техники».

Предназначены для студентов направления 09.04.01 Информатика и вычислительная техника.

Текст печатается в авторской редакции

Подпись в печать. Формат 60x84 1/16
Усл. печ. л. 1,28. Уч-изд. л. 1,16. Тираж 50 экз. Заказ 215 Бесплатно.
Юго-Западный государственный университет.
305040, г. Курск, ул. 50 лет Октября, 94.

Содержание

1 Общие положения

Цикл лабораторных работ по дисциплине «Современные проблемы информатики и вычислительной техники» выполняется студентами специальности Информатика и вычислительная техника, направленность «Элементы и устройства вычислительной техники и информационных систем» с целью закрепления и углубления полученных теоретических знаний, а также приобретения навыков проектирования структуры искусственных нейронных сетей и работы с ними посредством языка программирования Python, и фрэймворка Keras. В процессе выполнения лабораторных работ происходит развитие навыков самостоятельной учебной и исследовательской работы студентов.

Каждому студенту выдается индивидуальное задание на цикл лабораторных работ в соответствии с номером варианта.

2 Структура цикла лабораторных работ

Цикл лабораторных работ состоит из следующих этапов:

1. Получение варианта задания в соответствии с индивидуальным номером.
2. Лабораторная работа №1 «Настройки программного окружения для моделирования нейронной сети».
3. Лабораторная работа №2 «Нейросетевое распознавание изображений цифр».

Лабораторная работа №1 посвящена установке и настройке программного инструментария для моделирования искусственных нейронных сетей. Результатом работы является программная среда, подготовленная для нейросетевого распознавания тестовых наборов данных.

Лабораторная работа №2 посвящена созданию, обучению и исследованию нейронной сети глубокого обучения на тестовом наборе изображений рукописных цифр MNIST.

3 Защита лабораторных работ

Каждая лабораторная работа считается выполненной после исправления недостатков и ошибок, обнаруженных при проверке преподавателем. Функционирование итоговой нейронной сети демонстрируется преподавателю.

Защиты лабораторных работ проводятся в сроки, определенные программой по дисциплине, и являются обязательной формой проверки выполненной работы.

Результаты защиты лабораторных работ, оцениваются дифференциальной отметкой по шестибальной шкале. Основными критериями при выставлении оценки является:

1. Самостоятельность выполненной работы.
2. Теоретическая и практическая подготовка студента.
3. Навыки исследовательского характера.
4. Грамотность изложения и свободное владение материалом.
5. Качество оформления отчета по лабораторной работе.
6. Правильность ответов на вопросы при защите.

Порядок защиты является следующим:

1. Доклад студента о результатах работы (2-5 мин).
2. Ответы на вопросы.

Студент, не предоставивший в установленный срок все лабораторные работы или не защитивший их по неуважительной причине, не допускается к промежуточной аттестации.

4 Методические указания по выполнению лабораторных работ

Этапы выполнения работы.

1. Получить вариант задания в соответствии с индивидуальным номером.
2. Выполнить лабораторную работу №1 «Настройки программного окружения для моделирования нейронной сети».
3. Выполнить лабораторную работу №2 «Нейросетевое распознавание изображений цифр».
4. Оформить цикл лабораторных работ в виде единого отчета.

Базовые требования к выходной программе.

Функциональные требования:

1. выполнение условий, заданных индивидуальным вариантом;
2. отображение текущего состояния процесса обучения искусственной нейронной сети;
3. отображение результирующей точности распознавания на тестовой выборке.

Требования к условиям эксплуатации:

1. IBM-совместимая ПЭВМ.
2. Операционная система Linux(Ubuntu 16.04) / MS Windows (8.0 или выше).
3. Язык программирования Python (дистрибутив - Anaconda3).
4. Среда программирования PyCharm.

4.1 Теоретическая справка

Искусственные нейронные сети (ИНС) глубокого обучения в настоящее время становятся одним из самых популярных методов машинного обучения.

Искусственная нейронная сеть состоит из простых вычислительных элементов – искусственных нейронов, функционирующих по аналогии с нейронами головного мозга.

Работа традиционных алгоритмов машинного обучения существенно зависит от того, какие данные из общего объема будут выбраны для решения задачи. Такие данные называются признаками (features). Существует целое научное направление, посвященное выбору признаков, которое называется feature engineering. Однако до сих пор нет конструктивных рекомендаций по эффективному выбору признаков. Если для задачи получилось подобрать нужные признаки, то ее удается решить. В противном случае задача остается нерешенной.

Глубокие нейронные сети, в отличие от альтернативных подходов, могут работать со всем набором имеющихся данных. В процессе обучения нейронная сеть сама определяет, какие признаки в данных важны, а какие нет. Для обучения глубоких нейронных сетей требуются большие вычислительные ресурсы.

ИНС сейчас применяются для решения большого количества задач разного типа. Среди таких задач можно отметить:

- Оформление фото или видео в стиле изображений известных художников (мобильные приложения Prisma и Artisto);
- Сочинение музыки;
- Автоматический перевод текста на разных языках (Google Translate, Skype Translator, Яндекс.Переводчик);

• Самоуправляемые автомобили (Waymo, ранее Google Self-Driving Car, Яндекс.Такси, Uber);

• Робототехника и электронные помощники (Робот-консьерж в отеле Хилтон);

• Анализ медицинских изображений.

Популярность нейронных сетей в последнее время обусловлена следующими факторами:

1. Накопление существенного объема данных различного рода (фотографии, аудиозаписи, видеозаписи). Это позволяет обеспечить выборку данных, достаточного размера для тренировки нейронных сетей.

2. Производительность компьютеров резко возросла. Появились многоядерные процессоры и графические ускорители (GPU), которые по производительности превосходят суперкомпьютеры, доступные ученым в 80-е и 90-е годы прошлого века. Такие вычислительные мощности нужны для того, чтобы обучать глубокие нейросети на большом объеме данных.

3. Методы глубокого обучения нейронных сетей были усовершенствованы. В последнее время было предложено много небольших улучшений, таких как методы нормализованной инициализации весов, слои пакетной нормализации и многое другое.

4. Появилось большое количество готовых к использованию программ глубокого обучения нейронных сетей. Уже не требуется реализовывать глубокое обучение самостоятельно, можно взять готовую библиотеку и использовать ее для решения практических задач.

4.2 Настройки программного окружения для моделирования нейронной сети

Опишем процесс установки программного обеспечения, необходимого для проведения лабораторных работ. В комплект входит:

1. Anaconda3 – дистрибутив языка программирования Python; Рекомендуемая версия – предлагается сайтом разработчика по умолчанию.

2. Keras – открытая нейросетевая библиотека, написанная на языке Python. Представляет собой надстройку над фреймворками Deeplearning4j, TensorFlow и Theano. Рекомендуемая версия 2.0.9.

3. Theano – библиотека численного вычисления в Python. Рекомендуемая версия 0.9.0.

4. Visual studio – среда разработки и компилятор языка c++. Рекомендуемая версия Visual studio community edition 2015.

5. CUDA – программно-аппаратная архитектура параллельных вычислений, которая позволяет существенно увеличить вычислительную производительность благодаря использованию графических процессоров фирмы Nvidia. Рекомендуемая версия 8.0. Для установки может потребоваться учетная запись разработчика NVidia.

6. CuDNN – библиотека примитивов глубоких нейронных сетей для ускорения GPU-вычислений с использованием CUDA. Рекомендуемая версия 5.1.

7. PyCharm – среда разработки для языка программирования Python. Рекомендуемая версия – предлагается сайтом разработчика по умолчанию.

Загрузите и установите Anaconda. Не рекомендуется устанавливать дистрибутив в директорию по умолчанию, не все версии пакетов поддерживают корректную работу с путями, содержащими пробелы.

Загрузите дистрибутив Anaconda, работающий с Python 3.4:

«https://www.continuum.io/downloads»

Примечание: Обратите внимание на разрядность вашей операционной системы, она должна совпадать с разрядностью скачиваемых пакетов.

Примечание: Все команды, приведенные здесь и далее следует вводить в терминале "Anaconda Prompt", входящем в комплект поставки Anaconda, если не указано иное.

Создайте новое виртуальное окружение для своего проекта.

Примечание: Виртуальное окружение для Питона – очень удобный инструмент при одновременной работе с несколькими проектами. При разработке вы устанавливаете различные библиотеки, да и версия самого Python может отличаться.

Использование виртуального окружения позволяет абстрагироваться от библиотек, используемых в системе. Вы создаете виртуальное окружение для конкретного проекта, активируете его, и все устанавливаемые модули Питона будут установлены только в данном виртуальном окружении. Чтобы работать в другом проекте с другими версиями библиотек, достаточно просто переключить виртуальное окружение.

Параметр «python» служит для выбора используемой в проекте версии питона. Параметр «anaconda» служит для подключения к новой среде основных пакетов. Откройте "Anaconda Prompt" и введите приведенную ниже команду, указав желаемую версию:

```
conda create -n env_name35 anaconda python=3.5
```

Теперь вы находитесь в новом виртуальном окружении. Об этом говорит заголовок диалогового окна "Anaconda Prompt (env_name35)". Импортируйте пакет theano используя следующую команду:

```
conda install theano
```

Примечание: В дальнейшем для наглядного отображения прироста производительности от использования графического ускорителя можно воспользоваться специальным скриптом (check_blas.py), загрузить который можно по адресу:

https://raw.githubusercontent.com/Theano/Theano/master/theano/misc/check_blas.py

Запустите его из консоли Anaconda Prompt, предварительно перейдя в каталог, в который вы его сохранили:

```
python check_blas.py
```

Вы увидите сообщение схожего содержания:

```
Total execution time: 25.74s on CPU (without direct Theano binding to blas but with numpy/scipy binding to blas).
```

Для того, чтобы использовать BLAS в Theano создайте файл .theanorc в директории *C:\Users\[Your_User], где [Your_User]* имя вашего пользователя и добавьте в него следующие строки:

```
[global]
floatx = float32
[blas]
ldflags = -LC:\Anaconda3\Library\bin -lmkl_rt
```

Это позволит использовать библиотеку mklBLAS, входящую в основной комплект библиотек Anaconda, скопированный в наше новое окружение. Инструкции по установке обычно содержат рекомендации использовать OpenBLAS, но mklBLAS уже включен в ядро Anaconda, и дает порядка 30% прироста по скорости. Без использования библиотеки BLAS вы не сможете использовать некоторые слои, например, сверточные слои, и вы можете получить ошибки такого вида:

```
AssertionError: AbstractConv2d Theano optimization failed: there is no implementation available supporting the requested options. Did you exclude both "conv_dnn" and "conv_gemm" from the optimizer? If on GPU, is cuDNN available and does the GPU support it? If on CPU, do you have a BLAS library installed Theano can link against?
```

Это так же ограничит вычисления размерностью float32, что позволяет существенно увеличить скорость. Если вы теперь запустите скрипт check_blas.py вы заметите прирост скорости:

```
Total execution time: 11.77s on CPU (with direct Theano binding to blas).
```

Теперь мы должны установить компилятор «Microsoft C++». Загрузить его можно по ссылке:

«https://www.visualstudio.com/free-developer-offers/»

Загрузите «Visual studio community edition». Вам нужно установить: «Programming Languages/Visual C++/Common Tools for Visual C++ 2015». Содержимое остальных секций устанавливать не обязательно

Теперь необходимо загрузить CUDA с сайта NVidia:

https://developer.nvidia.com/cuda-downloads

Необходимо установить следующие компоненты: CUDA/development, visual studio integration, runtime.

Для использования технологии CUDA в своих проектах необходимо добавить в файл .theanorc следущие строки:

```
[nvcc]
flags=--cl-version=2015 -D_FORCE_INLINES
```

если не указать флаг cl-version можно столкнуться со следующей ошибкой:

nvcc fatal : nvcc cannot find a supported version of Microsoft Visual Studio. Only the versions 2010, 2012, and 2013 are supported

Если NVCC не сможет увидеть файлы Visual Studio и файлы библиотек в новом формате- появятся сообщения следующего содержания:

LINK : fatal error LNK1104: cannot open file stdio.h

LINK : fatal error LNK1104: cannot open file libucrt.lib

LINK : fatal error LNK1104: cannot open file uuid.lib

Необходимо добавить новую переменную среды INCLUDE, содержащую следующие пути:

C:\Program Files (x86)\Windows Kits\10\Include\10.0.14393.0\um;C:\Program Files (x86)\Windows Kits\10\Include\10.0.14393.0\ucrt

Кроме того, необходимо добавить переменную среды LIB, содержащую следующие пути:

C:\Program Files (x86)\Windows Kits\10\Lib\10.0.14393.0\ucrt\x64;C:\Program Files (x86)\Windows Kits\10\Lib\10.0.14393.0\um\x64

Примечание: В зависимости от версии программ пути могут незначительно различаться. Убедитесь, что используете пути, содержащие «10.0.НаивысшийНомер».

Так же убедитесь, что переменна PATH включает следующие пути:

C:\Program Files (x86)\Microsoft Visual Studio 14.0\VC\bin\amd64

C:\Anaconda3

C:\Anaconda3\Scripts

C:\Anaconda3\Library\bin

C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v8.0\bin

C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v8.0\libnvvp

Перезагрузите компьютер для применения изменений. Теперь, если вы запустите check_blas.py вы заметите прирост скорости:

Total execution time: 9.83s on CPU (with direct Theano binding to blas).

Теперь .theanorc должен выглядеть примерно так:

```
[global]
floatx = float32
cxx = C:\Anaconda3\envs\env_name35\Library\mingw-w64\bin\g++.exe
mode = FAST_RUN
[blas]
ldflags = -LC:\Anaconda3\Library\bin -lmkl_rt
[gcc]
```

```
cxxflags = -LC:\Anaconda3\envs\env_name35\Library\mingw-w64\include -LC:\Anaconda3\envs\env_name35\Library\mingw-w64\lib -lm
[nvcc]
flags=--cl-version=2015 -D_FORCE_INLINES
```

Используем только CPU. Если в вашем ПК отсутствует видеокарта фирмы NVIDIA, приведите свой файл .theanorc к следующему виду:

```
[global]
floatx = float32
cxx = C:\Anaconda3\envs\env_name35\Library\mingw-w64\bin\g++.exe
mode = FAST_RUN
force_device = True
device = cpu
```

Без использования флага force_device вы можете получить следующие ошибки:

ERROR (theano.sandbox.cuda): Failed to compile cuda_ndarray.cu: ('nvcc return status...

Если вы хотите использовать графический ускоритель обновите файл .theanorc:

```
[global]
floatx = float32
cxx = C:\Anaconda3\envs\env_name35\Library\mingw-w64\bin\g++.exe
mode = FAST_RUN
device = gpu
```

Теперь, если вы запустите файл check_blas.py вы увидите существенный прирост скорости:

Total execution time: 0.75s on GPU.

Примечание. Для увеличения производительности от использования технологии CUDA можно воспользоваться библиотекой cuDNN, разработанной для нативной поддержки нейросетей.

Чтобы использовать cuDNN необходимо зарегистрироваться как разработчик на сайте NVIDIA и скачать архив с файлами библиотеки cuDNN, которые необходимо распаковать в соответствующие директории C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v8.0 bin/include/lib directories. Для

того, чтобы использовать возможности cuDNN необходимо добавить соответствующие строчки в файл .theanorc:

```
[dnn]
enabled=True
```

Если этого не сделать, можно получить ошибки такого вида:

Using gpu device 0: GeForce GTX 1060 (CNMeM is disabled, CuDNN not available)

В случае корректного подключения cuDNN после запуска check_blas.py вы увидите сообщение следующего содержания:

Using gpu device 0: GeForce GTX 1060 (CNMeM is disabled, cuDNN 5105)

Для корректного выделения памяти используется библиотека CNMeM, рекомендуется выделять не больше 70% в случае, если видеокарта служит источником изображения для монитора. Внесите соответствующие изменения в файл .theanorc:

```
[lib]
cnmem=0.70
```

Теперь после запуска какого-либо скрипта можно увидеть сообщение следующего содержания:

Using gpu device 0: GeForce GTX 1060 (CNMeM is enabled with initial size: 70.0% of memory, cuDNN 5105)

Примечание: Если вы включите CNMeM вам не будет нужен раздел DNN поскольку он будет включен автоамтически. 3 опции доступны для CNMeM:

0 – не используется;

0 < N <= 1: использовать N частей от памяти GPU (ограничено драйвером до значения .95) [Примечание: значение должно быть дробным, в промежутке от 0.25 до 1.0];

> 1 использовать эти значения для выделения памяти в МБ;

Если процент установлен слишком высоко можно получить следующую ошибку:

ERROR (theano.sandbox.cuda): ERROR: Not using GPU. Initialisation of device gpu

```
failed: initCnmem: cnmemInit call failed! Reason=CNMEM_STATUS_OUT_OF_MEMORY. numdev=1
```

Установка среды разработки PyCharm

Загрузите установочный файл с сайта разработчиков

https://www.jetbrains.com/pycharm/download/?gclid=CIaQwY2q8tACFZ26wAod-ScPwA&gclsrc=aw.ds.ds&dclid=CJC7042q8tACFRO6TwodaKAAAA#section=windows

Откройте PyCharm, на панели управления выберите «File», затем выберите «Configure/Settings», а затем «Project Interpreter». Нажмите на стрелку рядом с Project Interpreter и выберите «Show All». Нажмите на «+», затем «Add Local». Перейдите в директорию своей виртуальной среды и выберите python.exe:

C:\Anaconda3\envs\env_name35\python.exe

Установите keras из консоли «Anaconda Prompt» и обновите все библиотеки с помощью следующей команды:

conda upgrade --all

Итоговый файл .theanorc для использования GPU должен выглядеть примерно так:

```
[global]
floatx = float32
device = gpu
mode = FAST_RUN
cxx = C:\Anaconda3\envs\env_name35\Library\mingw-w64\bin\g++.exe
[blas]
ldflags = -LC:\Anaconda3\Library\bin -lcublas64_80
[gcc]
cxxflags = -LC:\Anaconda3\envs\env_name35\Library\mingw-w64\include -LC:\Anaconda3\envs\env_name35\Library\mingw-w64\lib -lm
```

```
[nvcc]
flags=--cl-version=2015 -D_FORCE_INLINES
[dnn]
enabled=True
[lib]
cnmem=0.70
```

Для использования CPU файл .theanorc выглядит примерно следующим образом:

```
[global]
floatx = float32
force_device = True
device = cpu
mode = FAST_RUN
cxx = C:\Anaconda3\envs\env_name35\Library\mingw-w64\bin\g++.exe
[blas]
ldflags = -LC:\Anaconda3\Library\bin -lmkl_rt
[gcc]
cxxflags = -LC:\Anaconda3\envs\env_name35\Library\mingw-w64\include -LC:\Anaconda3\envs\env_name35\Library\mingw-w64\lib -lm
[nvcc]
flags=--cl-version=2015 -D_FORCE_INLINES
```

4.3 Нейросетевое распознавание изображений цифр

Реализуем простейшую глубокую нейронную сеть – многослойный персептрон с двумя скрытыми слоями – и применим ее к задаче распознавания рукописных цифр из набора данных MNIST.

Необходимы следующие импорты:

```
from keras.datasets import mnist # утилиты для загрузки набора данных
from keras.models import Model # основной класс для инициализации и тренировки модели
from keras.layers import Input, Dense # два типа слоев нейронной сети, которые мы будем использовать
from keras.utils import np_utils # вспомогательные утилиты
```

Определим некоторые параметры нашей модели. Эти параметры часто называют гиперпараметрами, так как предполагается, что их значения будут заданы еще до начала обучения. Возьмем заранее подобранные значения:

batch_size – количество обучающих образцов, обрабатываемых одновременно за одну итерацию алгоритма градиентного спуска;

num_epochs – количество итераций обучающего алгоритма по всему обучающему множеству;

hidden_size – количество нейронов в каждом из двух скрытых слоев нейронной сети.

```
batch_size = 128 # на каждой итерации мы рассматриваем 128 тренировочных
примеров за раз
num_epochs = 20 # мы проходим весь тренировочный набор 20 раз
hidden_size = 512 # каждый скрытый слой содержит 512 нейронов
```

Загрузим тренировочный набор MNIST и проведем предварительную обработку. С помощью Keras это делается достаточно просто: он считывает данные с удаленного сервера напрямую в массивы библиотеки NumPy.

Чтобы подготовить данные, сначала мы представим изображения в виде одномерных массивов (так как считаем каждый пиксель отдельным входным признаком), а затем разделим значение интенсивности каждого пикселя на 255, чтобы новое значение попадало в отрезок [0, 1]. Это простой способ нормализовать данные.

Хорошим подходом к задаче классификации является вероятностная классификация, при которой имеется один выходной нейрон для каждого класса, выдающий вероятность того, что входной элемент принадлежит данному классу. Это подразумевает необходимость преобразования обучающих выходных данных в прямое кодирование: например, если желаемый выходной класс – 3, а всего классов пять (и они пронумерованы от 0 до 4), то подходящее прямое кодирование – [0, 0, 0, 1, 0]. Keras предлагает нам всю эту функциональность “из коробки”.

```
num_train = 60000 # 60000 тренировочных примеров из MNIST
num_test = 10000 # 10000 тестовых примеров из MNIST

height, width, depth = 28, 28, 1 # изображения в MNIST имеют размерность 28x28 и
цветовое пространство «градации серого»
num_classes = 10 # 10 классов(по 1 на каждую цифру)

(X_train, y_train), (X_test, y_test) = mnist.load_data() # загрузим данные MNIST

X_train = X_train.reshape(num_train, height * width) # Представим данные в виде
одномерных массивов
```

```
X_test = X_test.reshape(num_test, height * width) # Представим данные в виде
одномерных массивов D
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
X_train /= 255 # нормализуем данные до диапазона [0, 1]
X_test /= 255 # нормализуем данные до диапазона [0, 1]


Y_train = np_utils.to_categorical(y_train, num_classes) # конвертируем отметки
значений в прямое кодирование
Y_test = np_utils.to_categorical(y_test, num_classes) # конвертируем отметки
значений в прямое кодирование
```

Для определения модель воспользуемся стеком из трех Dense слоев, который соответствует полносвязному многослойному персептрону, где все выходы одного слоя связаны со всеми входами последующего. Будем использовать ReLU для нейронов первых двух слоев, и softmax для последнего слоя. Эта функция активации разработана, чтобы превратить любой вектор с реальными значениями в вектор значений вероятности и определяется для j-го нейрона следующим образом:

$$\sigma(\vec{z})_j = \frac{exp(z_j)}{\sum_i exp(z_i)}$$

.

Замечательная черта Keras, которая отличает его от других фреймворков (например, от TensorFlow) – это автоматический расчет размеров слоев; достаточно только указать размерность входного слоя, а Keras автоматически проинициализирует все остальные слои. Когда все слои определены, нужно просто задать входные и выходные данные, как это сделано ниже.

```
inp = Input(shape=(height * width,)) # входной одномерный вектор размерностью
784
hidden_1 = Dense(hidden_size, activation='relu')(inp) # Первый скрытый слой с
активационной функцией ReLU
hidden_2 = Dense(hidden_size, activation='relu')(hidden_1) # Втророй скрытый
слой с активационной функцией ReLU
out = Dense(num_classes, activation='softmax')(hidden_2) # Output softmax layer

model = Model(input=inp, output=out) # To define a model, just specify its input
and output layers
```

Теперь осталось только определить функцию потерь, алгоритм оптимизации и метрики, которые следует собрать.

В случае использования вероятностной классификации, в качестве функции потерь лучше всего использовать не квадратичную ошибку, а перекрестную энтропию. Для определенного выходного вероятностного

вектора $\vec{y}$, сравниваемого с фактическим вектором $\vec{\hat{y}}$, потеря (для k-го класса) будет определяться как

$$\mathcal{L}(\vec{y},\vec{\hat{y}}) = -\sum_{i=1}^{k} \hat{y}_i \log y_i$$

.

Потери будут меньше для вероятностных задач (например, с логистической/softmax функцией для выходного слоя), в основном из-за того, что данная функция предназначена для максимизации уверенности модели в правильном определении класса, и на нее не влияет распределение вероятностей попадания образца в другие классы (в то время как функция квадратичной ошибки стремится к тому, чтобы вероятность попадания в остальные классы была как можно ближе к нулю).

Используемый алгоритм оптимизации будет напоминать какую-то форму алгоритма градиентного спуска, отличие будет лишь в том, как выбирается темп обучения η. Используем оптимизатор Адама, который обычно показывает хорошую производительность.

Так как классы сбалансированы (количество рукописных цифр, принадлежащих каждому классу, одинаково), подходящей метрикой будет точность (accuracy) – доля входных данных, отнесенных к правильному классу.

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

Наконец, запускаем обучающий алгоритм. Хорошей практикой будет отложить некоторое подмножество данных для проверки, что алгоритм верно распознает данные – эти данные еще называют валидационным набором (validation set); отделим для этой цели 10% данных.

Еще одна приятная особенность Keras – детализация: он выводит детальное логирование всех шагов алгоритма.

```
model.fit(X_train, Y_train, # тренируем сеть используя тренировочный набор.
          batch_size=batch_size, nb_epoch=num_epochs,
          verbose=1, validation_split=0.1) # отделим 10% для тестовой выборки
model.evaluate(X_test, Y_test, verbose=1) # Оценим подготовленную модель на
тестовом наборе!
```

Результат работы:

```
Train on 54000 samples, validate on 6000 samples
Epoch 1/20

  128/54000 [..............................] - ETA: 7s - loss: 2.3805 - acc: 0.0703
 4224/54000 [=>............................] - ETA: 0s - loss: 0.7891 - acc: 0.7701
……
53248/54000 [============================>.] - ETA: 0s - loss: 0.0091 - acc: 0.9973
54000/54000 [==============================] - 1s 11us/step - loss: 0.0090 - acc: 0.9973 - val_loss: 0.0985 - val_acc: 0.9832
```

Как видно, представленная модель достигает точности приблизительно 98.3% на тестовом наборе данных, это вполне достойно для такой простой модели, несмотря на то, что ее далеко превзошли сверхсовременные подходы.

Поэкспериментируйте с этой моделью: попробуйте различные гиперпараметры, алгоритмы оптимизации, функции активации, добавьте больше скрытых слоев, для того, чтобы достичь точности свыше 99%.

Библиографический список

1. Комашинский В.И., Смирнов Д.А. Глава 17. Нейронные сети и их применение в системах управления и связи. – М.: «Горячая линия-Телеком», 2003. – С. 94.
2. Antonio Gulli, Sujit Pal. Deep Learning with Keras – М.: «Вильямс», 2011. – 1248 с.
3. Николенко С., Кадурин А., Архангельская Е. Глубокое обучение. – СПб: Питер, 2017. – 480с.
4. Рашка С. Python и машинное обучение. – М.: ДМК Пресс, 2017. – 418с.