МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра математического обеспечения и применения ЭВМ

ОТЧЕТ

по лабораторной работе №4

по дисциплине «Искусственные нейронные сети»

Тема: «Распознавание рукописных символов»

Студент гр. 8383 _____ Муковский Д.В.

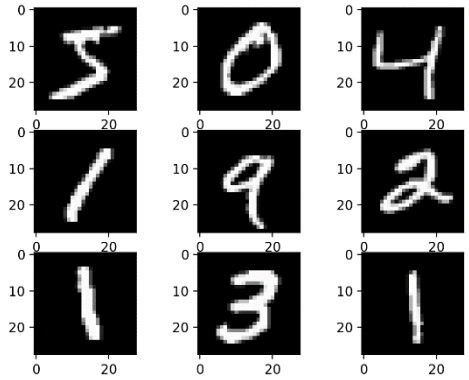
Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург

2021

Цель работы

Реализовать классификацию черно-белых изображений рукописных цифр (28x28) по 10 категориям (от 0 до 9).



Набор данных содержит 60,000 изображений для обучения и 10,000 изображений для тестирования.

Задачи

- Ознакомиться с представлением графических данных
- Ознакомиться с простейшим способом передачи графических данных нейронной сети
- Создать модель
- Настроить параметры обучения
- Написать функцию, позволяющая загружать изображение пользователи и классифицировать его

Требования

- Найти архитектуру сети, при которой точность классификации будет не менее 95%
- Исследовать влияние различных оптимизаторов, а также их параметров, на процесс обучения

- Написать функцию, которая позволит загружать пользовательское изображение не из датасета

Ход работы

Исходная архитектура ИНС, представленная в задании, была протестирована и показала точность в 97%, значит ее мы трогать не будем

Далее были исследовано влияние разных оптимизаторов, а также их параметров на процесс обучения.

SGD

Стохастический оптимизатор градиентного спуска.

Включает в себя поддержку импульса, затухания скорости обучения и импульса Нестерова.

Аргументы

- **learning_rate:** float>= 0. Скорость обучения.
- **momentum:** float>= 0. Параметр, ускоряющий SGD в соответствующем направлении и гасящий колебания.
- **nesterov:** boolean. Применять ли импульс Нестерова?

Результаты в зависимости от параметров

`learning_rate=0.01, momentum=0.0, nesterov=False`	`test_loss: 0.329` `test_acc: 90.93%`
`learning_rate=0.1, momentum=0.0, nesterov=False`	`test_loss: 0.138` `test_acc: 95.91%`
`learning_rate=0.1, momentum=0.5, nesterov=False`	`test_loss: 0.096` `test_acc: 96.92%`
`learning_rate=0.1, momentum=0.5, nesterov=True`	`test_loss: 0.103` `test_acc: 96.94%`

RMSprop

Рекомендуется оставить параметры этого оптимизатора на значениях по умолчанию (за исключением скорости обучения, которая может быть свободно настроена).

Аргументы

- **learning_rate**: float >= 0. Скорость обучения.
- **rho**: float >= 0.

Результаты в зависимости от параметров

learning_rate=0.001, rho=0.9	test_loss: 0.072 test_acc: 97.75%
learning_rate=0.01, rho=0.9	test_loss: 0.187 test_acc: 97.01%
learning_rate=0.1, rho=0.9	test_loss: 0.756 test_acc: 89.32%
learning_rate=0.001, rho=0.5	test_loss: 0.086 test_acc: 97.53%

Adagard

Адаград — это оптимизатор, в котором скорость обучения зависит от конкретных параметров, которые адаптированы к тому, как часто параметр обновляется во время обучения. Чем больше обновлений получает параметр, тем меньше скорость обучения.

Рекомендуется оставлять параметры этого оптимизатора на значениях по умолчанию.

Аргументы

learning_rate: float >= 0. Уровень начального обучения.

Результаты в зависимости от параметров

learning_rate=0.01	test_loss: 0.235 test_acc: 93.45%
learning_rate=0.001	test_loss: 0.542 test_acc: 87.24%
learning_rate=0.1	test_loss: 0.082 test_acc: 97.53%

Adadelta

Adadelta — более надежное расширение Adagrad, которое адаптирует скорость обучения на основе скользящего окна обновления градиентов вместо того, чтобы накапливать все градиенты прошлых лет. Таким образом, Adadelta продолжает обучение даже тогда, когда сделано много обновлений. По сравнению с Adagrad, в оригинальной версии Adadelta нет необходимости устанавливать начальную скорость обучения. В этой версии, как и в большинстве других оптимизаторов Keras, можно установить начальную скорость обучения и коэффициент распада.

Рекомендуется оставить параметры этого оптимизатора в их значениях по умолчанию.

Аргументы

learning_rate: float >= 0. Начальная скорость обучения, по умолчанию 1. Рекомендуется оставить значение по умолчанию.

rho: float >= 0. Коэффициент распада ададелькы, соответствующий доле градиента, который необходимо сохранять на каждом шаге.

Результаты в зависимости от параметров

learning_rate=1.0, rho=0.95	test_loss: 0.078 test_acc: 97.53%
learning_rate=1.0, rho=0.5	test_loss: 0.099 test_acc: 97.13%
learning_rate=2.0, rho=0.95	test_loss: 0.074 test_acc: 97.77%

Adam

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

Аргументы

learning_rate: float >= 0. Скорость обучения.

beta_1: float, 0 < beta < 1. Обычно близко к 1.

beta_2: float, 0 < beta < 1. Обычно близко к 1.

amsgrad: boolean. Применять ли AMSGrad

Результаты в зависимости от параметров

learning_rate=0.001, beta_1=0.9, beta_2=0.999, amsgrad=False	test_loss: 0.071 test_acc: 97.79%
learning_rate=0.01, beta_1=0.9, beta_2=0.999, amsgrad=False	test_loss: 0.127 test_acc: 96.74%
learning_rate=0.001, beta_1=0.95, beta_2=0.999, amsgrad=False	test_loss: 0.08 test_acc: 97.53%
learning_rate=0.001, beta_1=0.9, beta_2=0.999, amsgrad=True	test_loss: 0.077 test_acc: 97.46%
learning_rate=0.001, beta_1=0.9, beta_2=0.9, amsgrad=False	test_loss: 0.067 test_acc: 98.01%

Adamax

Это вариант Adam, основанный на норме бесконечности.

Аргументы

learning_rate: float >= 0. Скорость обучения.

beta_1: float, 0 < beta < 1. Обычно близко к 1.

beta_2: float, 0 < beta < 1. Обычно близко к 1.

Результаты в зависимости от параметров

learning_rate=0.001, beta_1=0.9, beta_2=0.999	test_loss: 0.086 test_acc: 97.33%
learning_rate=0.01, beta_1=0.9, beta_2=0.999	test_loss: 0.82 test_acc: 97.52%
learning_rate=0.1, beta_1=0.9, beta_2=0.999	test_loss: 0.14 test_acc: 96.01%
learning_rate=0.001, beta_1=0.99, beta_2=0.999	test_loss: 0.119 test_acc: 96.53%
learning_rate=0.001, beta_1=0.9, beta_2=0.9	test_loss: 0.073 test_acc: 97.76%

Nadam

Так же, как Adam, по сути, являестя RMSprop с импульсом, так и Nadam это RMSprop с импульсом Нестерова.

Аргументы

learning_rate: float >= 0. Скорость обучения.

beta_1: float, 0 < beta < 1. Обычно близко к 1.

beta_2: float, 0 < beta < 1. Обычно близко к 1.

Результаты в зависимости от параметров

`learning_rate=0.001, beta_1=0.9, beta_2=0.999`	`test_loss: 0.072` `test_acc: 97.77%`
`learning_rate=0.01, beta_1=0.9, beta_2=0.999`	`test_loss: 0.122` `test_acc: 97.23%`
`learning_rate=0.001, beta_1=0.99, beta_2=0.999`	`test_loss: 0.070` `test_acc: 97.76%`
`learning_rate=0.001, beta_1=0.999, beta_2=0.999`	`test_loss: 0.082` `test_acc: 97.46%`
`learning_rate=0.001, beta_1=0.99, beta_2=0.9`	`test_loss: 0.075` `test_acc: 97.61%`

Лучший результат точности показала модель с оптимизатором Adam, конфигурация: `learning_rate=0.001, beta_1=0.9, beta_2=0.9, amsgrad=False`, точность равна 98.01%.

Далее была написана функция (см. Приложение А) для загрузки пользовательских изображений и предсказывания какая цифра нарисована.

Пример работы

Картинки подобраны разных размеров

Результат выполнения работы на данных картинках:

```
File: number_1.png - prediction: 1
File: number_2.png - prediction: 2
File: number_3.png - prediction: 3
File: number_5.png - prediction: 5
File: number_6.png - prediction: 6
```

Выводы

В ходе выполнения лабораторной работы была написана ИНС, которая классифицирует черно-белые изображения рукописных цифр по 10 категориям. Была найдена оптимальная конфигурация оптимизатора сети, а также была написана функция считывания пользовательского изображения и представления его в виде тензора.

ПРИЛОЖЕНИЕ А
ИСХЛДНЫЙ КОД

```
import tensorflow as tf
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.models import Sequential
from tensorflow.keras.utils import to_categorical
from tensorflow.keras import optimizers
from tensorflow.keras.preprocessing.image import load_img
from tensorflow.keras.preprocessing.image import img_to_array
import numpy as np

mnist = tf.keras.datasets.mnist
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()

train_images = train_images / 255.0
test_images = test_images / 255.0

train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)

model = Sequential()
model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dense(10, activation='softmax'))

optimizer = optimizers.Adam(learning_rate=0.001, beta_1=0.9, beta_2=0.9)

model.compile(optimizer=optimizer, loss='categorical_crossentropy',
              metrics=['accuracy'])
model.fit(train_images, train_labels, epochs=5, batch_size=128)

test_loss, test_acc = model.evaluate(test_images, test_labels)
print('test_loss: ', test_loss)
print('test_acc: ', test_acc)
```

```
def predict_image(model, filename):
    image = load_img(filename, color_mode='grayscale', target_size=(28, 28))
    image = img_to_array(image).reshape(1, 28, 28, 1)
    image = image.astype('float32') / 255.0
    prediction = model.predict(image)
    print(f'File: {filename} - prediction: {np.argmax(prediction)}')


predict_image(model, 'number_1.png')
predict_image(model, 'number_2.png')
predict_image(model, 'number_3.png')
predict_image(model, 'number_5.png')
predict_image(model, 'number_6.png')
```