Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет Компьютерных сетей и систем

Обработка больших объемов информации

ОТЧЕТ

ПО ЛАБОРАТОРНОЙ РАБОТЕ 1

по предмету «Интеллектуальный анализ информации»

Студент
гр. 758601
Лимонтов А. С.

Проверил
Ивашенко В. П.

Постановка задачи

Реализация линейной рециркуляционной сети как модели автодекодера для задачи понижения размерности данных.

Этапы реализации программы

Исходные данные

Импорты используемых библиотек:

```
import numpy as np
import random
from PIL import Image
from matplotlib.pyplot import imshow
from functools import partial
from tqdm import tqdm
import matplotlib.pyplot as plt
from os import walk
from itertools import tee

%matplotlib inline
```

Задание глобальных параметров:

- `SEED`: для случайных генераторов с целью воспроизводимости
- `MODE`: режим работы с картинкой RGB/L - цветной/черно-белый
- `S`: количество аргументов на пиксель (3** - для цветного изображения, **1 - для черно-белого)
- `IMG_PATH`: путь к обрабатываемому изображению

```
SEED = 342323425
np.random.seed(SEED)
MODE = 'L'
S = 1
IMG_PATH = './rsz_pic3.jpg'
```

Метод загрузки картинки и преобразования в необходимый режим и размер:

```
def load_image(path):
    img = Image.open(path)
    img = img.resize((128, 128))
    return img.convert(MODE)
```

Обработка изображения

Методы преобразования картинки, генераторы нарезаемых блоков. Картинка равномерно нарезается на блоки последовательно слева-направо и сверху-вниз на размеры nxm. При декодировании следует помнить о том, что в результате должны получиться

- целый значения
- числа в пределах [0..255]

Поэтому делается `np.clip(np.int_(C_MAX * (area + 1.0) / 2.0), 0, 255:`

```
def encode_pixel(P):
    C_MAX = 255.0
    P = np.array(P)
    return (2.0 * P / C_MAX) - 1.0

def encode_area(im, lu_corner, rb_corner):
    X0, Y0 = lu_corner
    X1, Y1 = rb_corner

    def transform(x, y):
        return encode_pixel(im.getpixel((x, y)))

    q = []
    for x in range(X0, X1 + 1):
        for y in range(Y0, Y1 + 1):
            q.append(transform(x, y))
    return np.array(q).flatten()

def decode_area(area):
    C_MAX = 255.0
    res = np.clip(np.int_(C_MAX * (area + 1.0) / 2.0), 0, 255)
    return res

def decode_image(areas):
    if S == 1:
        im_area = np.empty((H, W))
    else:
        im_area = np.empty((H, W, S))
    for i in range(H):
        for j in range(W):
            ii = i / n
            jj = j / m
            idx = ii * (W / m) + jj
```

```
            im_area[j][i] = decode_area(areas[idx][0][(i % n) * m + j % m:(i % n) *
m + j % m + S])
    return Image.fromarray(im_area.astype('uint8'), mode=MODE)

def transform_random_image(im, n, m):

    def transform(i, j):
        lu = (i, j)
        rb = (i + n - 1, j + m - 1)
        return encode_area(im, lu, rb)

    while True:
        i = np.random.randint(im.height - n)
        j = np.random.randint(im.width - m)
        X = transform(i, j)
        yield X.reshape(1, len(X))


def transform_image(im, n, m):

    def transform(i, j):
        lu = (i, j)
        rb = (i + n - 1, j + m - 1)
        return encode_area(im, lu, rb)

    for i in range(0, im.height, n):
        for j in range(0, im.width, m):
            X = transform(i, j)
            yield X.reshape(1, len(X))
```

Класс автокодировщика

Логика автокодировщика. Здесь используется метод инициализации весов glorot_uniform (https://keras.io/initializers/#glorot_uniform and http://proceedings.mlr.press/v9/glorot10a/glorot10a.pdf). На вход можно подавать следующие параметры:

- `input_layers`: количество нейронов на входном слое (равно размеру вектора, вычисляемого из нарезанного блока)
- `lr`: learning rate
- `dynamic_lr`: флаг, отвечающий за динамический пересчет learning rate после каждой итерации, либо использовать переданное значение
- `z`: коэффициент сжатия. Из него вычисляется количество нейронов на втором слое (int(z * input_layers))
- `w_norm`: флаг, отвечающий за нормировку весов после каждой итерации

Вызов автокодировщика прогоняет все блоки картинки (выполняет одну эпоху). Каждую эпоху насчитывается суммарная ошибка на всех обработанных блоках.

```
def glorot_uniform(input_layers, output_layers):
    limit = np.sqrt(6.0 / (input_layers + output_layers))
    return partial(np.random.uniform, low=-limit, high=limit)

class Autoencoder(object):
```

```
    def __init__(self, input_layers=1000, lr=1e-2, dynamic_lr=True, z=0.9,
w_norm=True):
        self.input_layers = input_layers
        self.mid_layers = int(z * input_layers)
        self.initializer = glorot_uniform(self.input_layers, self.mid_layers)
        self.lr = lr
        self.w_norm = w_norm
        self.dynamic_lr = dynamic_lr
        self.build()

    def error(self, err):
        return err**2

    def build(self):
        self.W1 = self.initializer(size=[self.input_layers, self.mid_layers])
        self.W2 = self.W1.T

    def __call__(self, gen, pbar=None):
        err = 0.0
        for inp in gen:
            mid, res = self.forward(inp)
            diff = res - inp
            err += self.error(diff).sum()
            self.backward(inp, mid, diff)
            if pbar:
                pbar.update(1)
        return err

    def forward(self, inp):
        mid = self.encode(inp, self.W1)
        return mid, self.decode(mid, self.W2)

    def backward(self, inp, mid, err):
        lr = self.lr
        if self.dynamic_lr:
            lr = 1.0 / np.dot(inp, inp.T)
        self.W1 -= lr * np.dot(np.dot(inp.T, err), self.W2.T)
        if self.dynamic_lr:
            lr = 1.0 / np.dot(mid, mid.T)
        self.W2 -= lr * np.dot(mid.T, err)

        if self.w_norm:
            self.W2 /= np.linalg.norm(self.W2, axis=0, keepdims=True)
            self.W1 /= np.linalg.norm(self.W1, axis=1, keepdims=True)

    def encode(self, inp, W1):
        return np.dot(inp, W1)

    def decode(self, mid, W2):
        return np.dot(mid, W2)
```

Тестирование модели

Метод для запуска обучения для выбранной модели. Контролируются следующие параметры:

- `model`: настроенная модель
- `epochs`: ограничение сверху на количество запускаемых эпох
- `name`: используется для отрисовки названия графика "Количество эпох на ошибку"
- `th_err`: пороговое значение для ошибки
- `viz`: флаг, нужно ли рисовать график
- `ax`: переданный объект plot, на котором нужно рисовать график (следует передать `viz=True`)
- `gen`: генератор для блоков. По умолчанию генерируются блоки из стандартной картинки размера n x m
- `progress`: progress bar для визуализации прогресса
- `L`: количество блоков (используется для progress bar по умолчанию)

Критерий останова: ошибка меньше пороговой либо количество запущенных эпох достигло значения epochs. В результат сохраняется следующая информация:

- `W1, W2`: веса на первом и втором слое соответственно
- `min_err`: минимальная достигнутая ошибка
- `epoch`: количество прошедших эпох (может быть меньше, чем epochs, если была получена ошибка меньше заявленной пороговой, но не может быть больше epochs)

```
def try_run(model, epochs, name='', th_err=0.1, viz=True, ax=None, gen=None, progress=True, L=L):
    np.random.seed(SEED)
    errors = []
    if gen is None:
        im = load_image(IMG_PATH)
        gen = transform_image(im, n, m)
    gens = tee(gen, epochs)
    best_results = {
        'W1': None,
        'W2': None,
        'min_err': np.inf,
        'epoch': epochs,
    }

    def run_iters(pbar=None):
        epoch_run = 0
        err = np.inf
        min_err = np.inf
        cur_gen = 0
        while epoch_run < epochs and err > th_err:
            t_gen = gens[cur_gen]
            cur_gen += 1
            errors.append(model(t_gen, pbar))
            if errors[-1] < min_err:
```

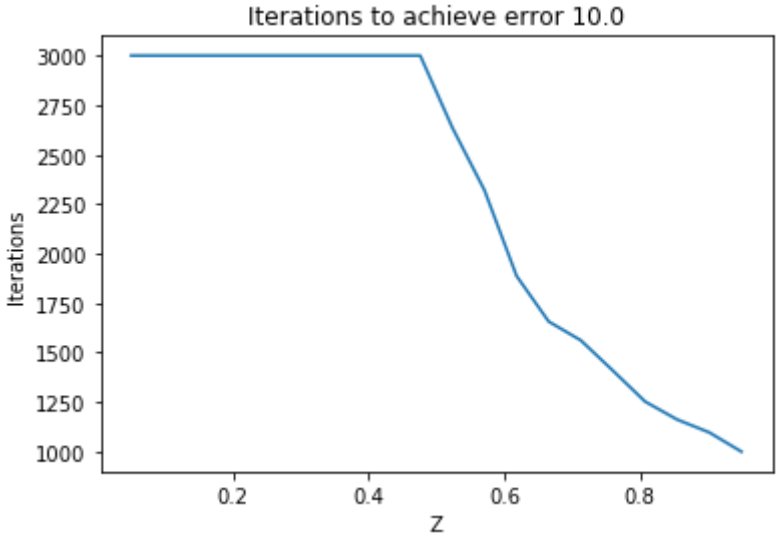
```
                best_results['min_err'] = errors[-1]
                best_results['W1'] = model.W1
                best_results['W2'] = model.W2
                min_err = errors[-1]
            err = errors[-1]
            epoch_run += 1

        best_results['epoch'] = epoch_run

    if progress:
        with tqdm(total=epochs*L) as pbar:
            run_iters(pbar)
    else:
        run_iters()

    if viz:
        epoch = best_results['epoch']
        x = np.arange(epoch)
        idx = np.argmin(errors)
        print("BEST ERROR {} at {}".format(errors[idx], idx))

        if ax is None:
            plt.plot(x, np.array(errors))
            plt.title(name)
            plt.plot(x[idx], errors[idx], 'rx--', linewidth=2, markersize=12)
            plt.show()
        else:
            ax.plot(x, np.array(errors))
            ax.set_title(name)
            ax.plot(x[idx], errors[idx], 'rx--', linewidth=2, markersize=12)
    return best_results

def viz_encode(model, best_results, path=IMG_PATH):
    W1 = best_results['W1']
    W2 = best_results['W2']
    min_err = best_results['min_err']
    epoch = best_results['epoch']

    print 'Showing results for min_err={}. Epoch run: {}'.format(min_err, epoch)
    areas = []
    im = load_image(path)
    for area in transform_image(im, n, m):
        areas.append(model.decode(model.encode(area, W1), W2))
    areas = np.array(areas)
    new_pic = decode_image(areas)
    return new_pic
```

Варьирование по количественным параметрам

Зависимость количества эпох от коэффициента сжатия Z. Фиксированные параметры:

- `th_err`: 10.0
- `epochs`: 3000

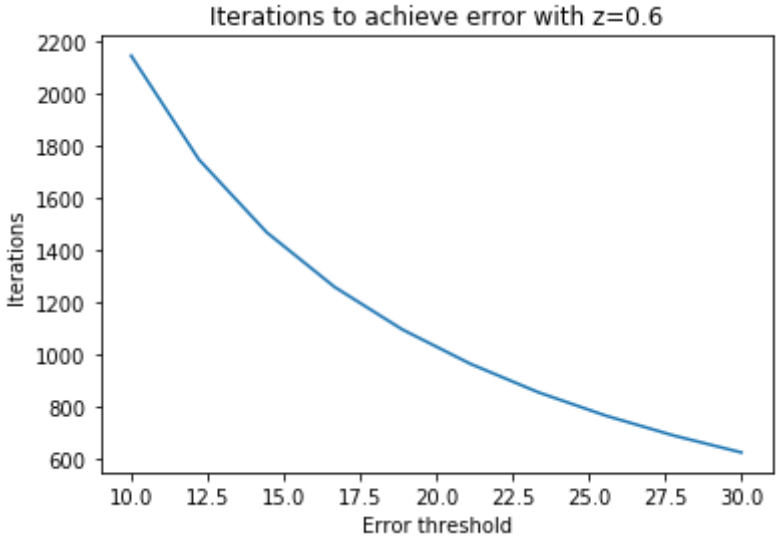
- `dynamic_lr`: True



Как видно из приведенного графика, при Z□0.5Z□0.5 количества эпох в размере 3000 не хватает, чтобы достичь ошибки в 10.0, и лишь для следующих коэффициентов это удается сделать. При этом сходится алгоритм значительно быстро - кривая очень наклонена

Построим график зависимости количества эпох от пороговой ошибки. Фиксированные параметры:

- `z`: 0.6
- `epoch`: 3000
- `dynamic_lr`: True

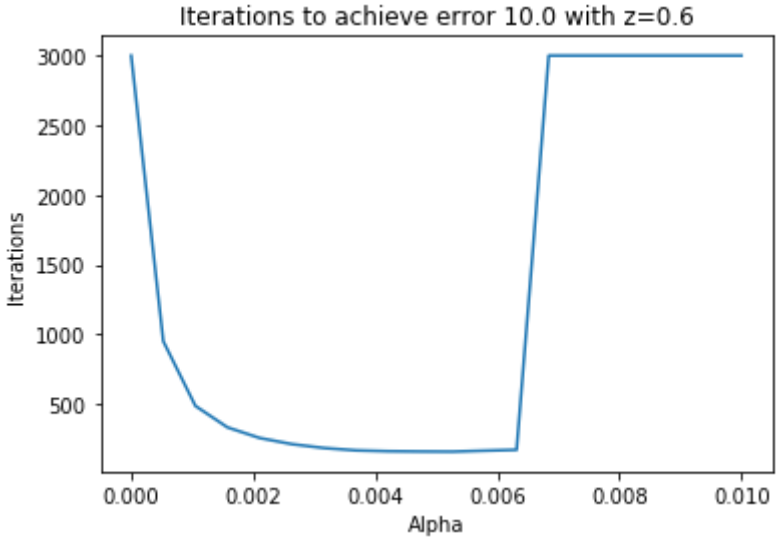


Как и ожидалось, количество затраченных эпох до достижения пороговой ошибки гладко снижается.

Зависимость количества эпох от коэффициента обучения (learning rate). Исходя из проведенных исследований, определим границы для него $[1e^{-6}..1e^{-2}]$. Фиксированные параметры:

- `z`: 0.6
- `th_err`: 10.0

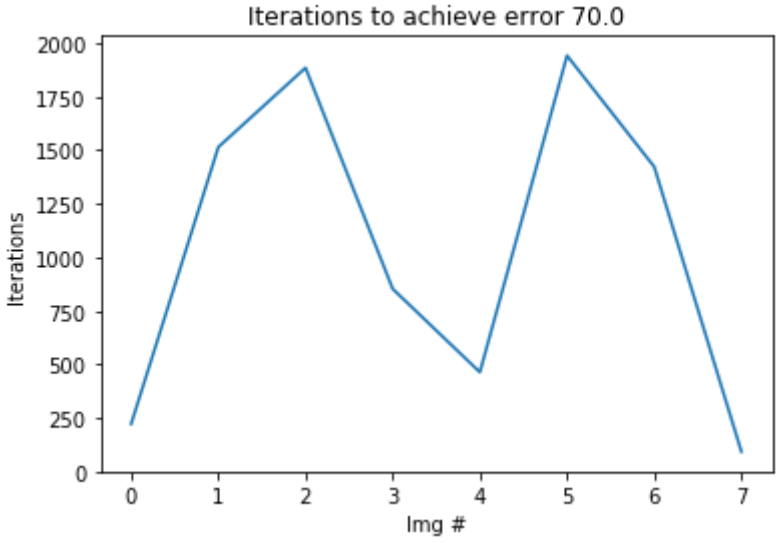
- `epoch`: 3000



Вывод по этому графику следующий: как и в других похожих алгоритмах обучения (например, в градиентном спуске) существуют две крайности задания learning rate (α): при слишком низком значении алгоритм "недообучается", то есть показывает плохой результат еще во время обучения, при слишком же большом значении алгоритм "промахивается" мимо глобальных минимумов. Похожая ситуация проявляется и здесь: при слишком маленьком αα алгоритму необходимо очень много времени (эпох), чтобы получить хорошую ошибку (график очень резко опускается, если идти слева), затем переходит в зону, в которой αα оптимально для обучения (как видно, это достигается где-то при α=[0.004..0.006], затем начинается резкий скачок, когда αα лишь делает только хуже всему процессу и выходит из зоны минимума

Зависимость количество эпох от картинки. Всего имеется 8 картинок, для них и будут проведены тесты. Фиксированные параметры:

- `z`: 0.6
- `epoch`: 3000
- `th_err`: 70.0
- `dynamic_lr`: True

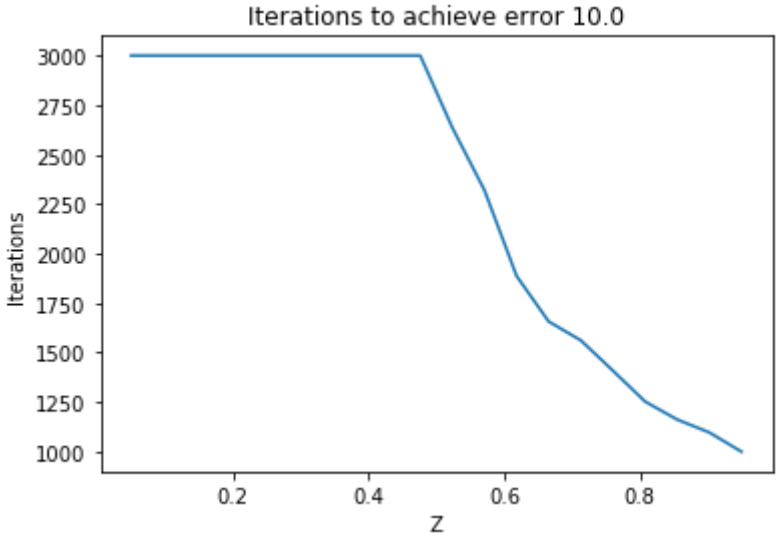


Видим, что в зависимости от содержания картинки (много однотонных областей, либо наоборот вся картинка "пестрит") алгоритм обучается неравномерно. Но в целом можно задать верхнюю планку на время обучения (в данном случае, где-то в районе 2000 эпох), для которой можно получить требуемую оценку в среднем для всех картинок.

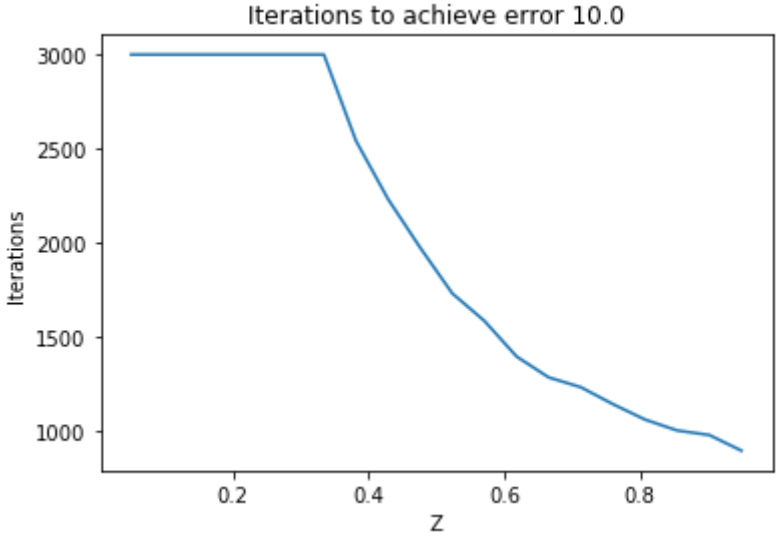
Варьирование по качественным параметрам

Теперь будем варьировать доступные параметры `w_norm` и `dynamic_lr`.

- `w_norm`: True
- `dynamic_lr`: True

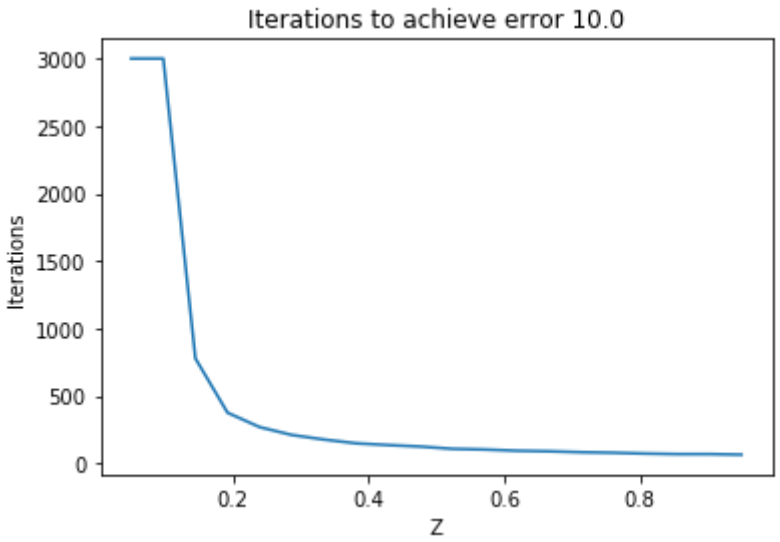


- `w_norm`: False
- `dynamic_lr`: True



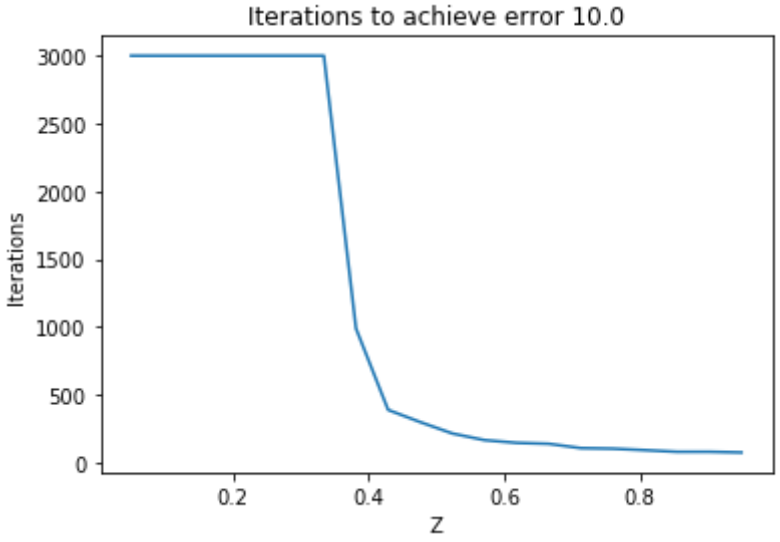
Без нормализации весов ошибка на небольших коэффициентах сжатия значительно лучше, чем со включенной нормализацией. Помимо прочего, сходимость на более высоких значениях Z также быстрее

- `w_norm`: False
- `dynamic_lr`: False (α=0.005)



С правильно подобранным αα результат становится еще лучше (быстрее сходится и в целом, и на малых Z)

- `w_norm`: True
- `dynamic_lr`: False (α=0.005)



С нормировкой весов и фиксированным αα наблюдается резкий скачок в районе Z≈0.4 к быстрейшей сходимости.

Выводы

Реализована рециркуляционная сеть как модель автокодировщика для задачи понижения размерности данных. Получены следующие результаты:

- Алгоритм выполняет меньшее число итераций с увеличением минимальной ошибки
- Алгоритм быстрее сходится при хорошо подобранном шаге обучения (≈0.005). Адаптивный шаг также работает неплохо, но все же при правильно подобранном αα можно добиться лучших результатов/скорости

- Применение техник нормирования и адаптивного обучения сокращает число итераций не всегда помогает. Это, в целом, зависит и от начального задания матриц весов
- С уменьшением числа входов/выходов уменьшается время работы алгоритма

Литература

1. How does Keras compare to other Deep Learning frameworks like Tensor Flow, Theano, or Torch? [Электронный ресурс] Режим доступа: https://www.quora.com/How-does-Keras-compare-to-other-Deep-Learning-frameworks-like-Tensor-Flow-Theano-or-Torch. Дата доступа: 01.12.2018
2. Keras Intro [Электронный ресурс] Режим доступа: https://keras.io/. Дата доступа: 01.12.2018
3. Keras [Электронный ресурс] Режим доступа: https://en.wikipedia.org/wiki/Keras. Дата доступа: 01.12.2018
4. Tensorflow on Keras [Электронный ресурс] Режим доступа: https://www.tensorflow.org/guide/keras. Дата доступа: 01.12.2018
5. 7 Steps to Mastering Deep Learning with Keras [Электронный ресурс] Режим доступа: https://www.kdnuggets.com/2017/10/seven-steps-deep-learning-keras.html. Дата доступа: 01.12.2018