Реализация сети Хопфилда для восстановления образов

Структура проекта

Создайте новую папку с проектом. В этой папке создайте:

- папка `img` - в ней хранятся картинки для обучения и тестирования сети
  - папка `train` - для обучающих картинок
  - папка `test` - для тестовых картинок
- файл `main.py` - файл программы с реализацией сети Хопфилда

Необходимые библиотеки (*Python*):

- `numpy` (`pip install numpy`)
- `matplotlib` (`pip install matplotlib`)
- `PIL` (`pip install pillow`)

Этапы реализации

Реализуемая нами сеть Хопфилда будет обучена на распознавание зашумленных образов, представленных в виде картинок. В каком виде необходимо представить картинки? Пусть картинка имеет только 2 цвета: черный и белый. Т.к. сеть Хопфилда работает с биполярными образами (`1` и `-1`), то закодируем черный цвет числом `-1`, белый: `1`. Соответственно, картинка представляет собой матрицу, содержащую только значения `-1` и `1`.

Но, стоит обратить внимание, что входной слой сети Хопфилда представляет собой вектор. Поэтому матрицу картинки необходимо будет преобразовать в вектор. Размер входного слоя в таком случае будет равен произведению кол-ва строк и столбцов матрицы (т.е. если картинка размерами `130x100`, то размер входного слоя равен `13000`).

Для реализации сети Хопфилда необходимо создать класс и определить в нем два поля для хранения размера входного слоя сети и матрицы весов (содержит только `0`). Пример:

```
class HopfieldNetwork:
    def __init__(self, size):
        self._size = size                       # размер входного слоя
        self._weights = np.zeros((size, size)) # матрица весов
```

Далее необходимо реализовать два метода:

- обучение сети
- распознавание образа

Реализация алгоритма обучения сети

Пусть обучающие примеры представляют собой матрицы P_i. Матрицу весов обозначим как W. Для обучения сети необходимо:

- в цикле по каждому обучающему примеру (матрице P_i):
  - преобразовать матрицу P_i в вектор-столбец V
  - вычислить $S = V \cdot V^T$
  - вычислить $W = W + S$
- обнулить диагональ матрицы W

Для преобразоваия матрицы P_i в вектор-столбец V в *Python* можно использовать функцию *reshape*:

```
# P является numpy-массивом
P = P.reshape(-1, 1)
```

Для выполнения матричного умножения можно использовать функцию `dot` библиотеки `numpy`:

```
numpy.dot(P, P.T)    # P.T - транспонированный вектор-столбец P
```

Для обнуления диагонали матрицы W можно использовать функцию `fill_diagonal` библиотеки `numpy`:

```
numpy.fill_diagonal(self._weights, 0)
```

Псевдокод фукнции обучения сети:

```
def train(self, train_patterns:list):
    for P in train_patterns:
        # преобразуем pattern в вектор-столбец
        # выполняем матричное умножение
        # результат умножения суммируем с матрицей весов
    # обнуляем главную диагональ матрицы весов
```

Реализация алгоритма распознавания образа

Пусть тестовый образ представляет собой матрицу P. Для начала необходимо матрицу P преобразовать в вектор V. Сделать это можно вызвав функцию `flatten`:

```
V = P.flatten()
```

Обозначим размер вектора V как $size$. Тогда Задача распознавания заключается в переписывании вектора V (обновлении состоянии i-го нейрона) следующим образом:

$$V_i = W_i \cdot V$$

где $i = 0, size$.

Причем после каждого такого обновления необходимо применить биполярную функцию активации для приведения значений вектора V в биполярные:

$$f(x) = \begin{cases} 1, V_i > 0, \\ -1, V_i \leq 0. \end{cases}$$

Выполнять полное обновление вектора V необходимо до тех пор, пока сеть не сойдется к стабильному состоянию. Однако, такое состояние может и не наступить, поэтому можно просто выполнить произвольное количество итераций.

Псевдокод функции распознавания образа:

```
def predict(self, test_pattern, max_iterations=10):
    # преобразуем матрицу в вектор (flatten)
    for _ in range(max_iterations):
        for i in range(self._size):
            # обновляем состояние i-го нейрона
            # применяем функцию активации. Пример:
            V[i] = 1 if V[i] > 0 else -1
    return V
```

Дополнительно

Картинки можно нарисовать в *Paint*. Для теста достаточно создать картинку размером `10x10`.

Для считывания изображений из папок `train` и `test` и преобразования их к биполярной матрице можете использовать следующий код:

```
class Util:
    @staticmethod
    def normalize(img_arr:np.ndarray) -> np.ndarray:
        '''
        Преобразует матрицу к биполярному виду (нормализация)
        '''
        return np.where(img_arr > 127, 1, -1)
```

```
    @staticmethod
    def load_img_as_array(img_folder:str) -> np.ndarray:
        '''
        Загружает картинки из заданной папки и нормализует их.
        Возвращает массив, содержащий нормализованные матрицы.
        '''
        arr = []
        for e in os.listdir(img_folder):
            image = Image.open(f'{img_folder}\\{e}')
            image = image.convert('L')
            norm_image = Util.normalize(np.array(image))
            arr.append(norm_image)
        return arr
```

Пример использования этих функций:

```
TRAIN_PATH = os.getcwd() + '\\img\\train'
TEST_PATH = os.getcwd() + '\\img\\test'

train = Util.load_img_as_array(TRAIN_PATH)
test = Util.load_img_as_array(TEST_PATH)
```

Сохраните исходный размер картинок, чтобы в дальнейшем преобразовать вектор, получаемый в результате распознавания, в матрицу.

```
SIZE_X, SIZE_Y = train[0].shape
```

Создайте объект класса сети и обучите ее:

```
net = HopfieldNetwork(SIZE_X * SIZE_Y)
net.train(train)
```

Протестируйте обученную сеть на тестовых образах:

```
results = []

for e in test:
        predicted = net.predict(e)
        # преобразуем вектор в матрицу
        predicted = predicted.reshape(SIZE_X, SIZE_Y)
        # сохраняем в виде кортежа: (тестовый образ, распознанный образ)
        results.append((e, predicted))
```

Для вывода полученных результатов можно использовать следующую функцию (добавьте ее в класс `Util`):

```
@staticmethod
def show_results(results:list) -> None:
        fig, axes = plt.subplots(len(results), 2)

        cmap = plt.cm.gray
        bounds = [-1.5, 0, 1.5]
        norm = plt.matplotlib.colors.BoundaryNorm(bounds, cmap.N)

        for i, e in enumerate(results):
                axes[i][0].imshow(e[0], cmap=cmap, norm=norm)
                axes[i][0].set_title('Test pattern')
                axes[i][0].axis('off')

                axes[i][1].imshow(e[1], cmap=cmap, norm=norm)
                axes[i][1].set_title('Predicted pattern')
                axes[i][1].axis('off')

        plt.tight_layout()
        plt.show()
```

И вызовите ее, передав массив с кортежами:

```
Util.show_results(results)
```