دانشگاه صنعتی امیرکبیر
(پلی تکنیک تهران)
دانشکده مهندسی برق

گزارش آزمایش ششم
مقدمه‌ای بر هوش محاسباتی
دکتر عبداللهی

نام و نام خانوادگی
کیان بهزاد

شماره دانشجویی
۹۵۲۳۰۱۷

پیاده سازی الگوریتم Hopfield

قصد داریم به پیاده سازی این الگوریتم در محیط **پایتون** بپردازیم.
پس در ابتدا می‌بایست توابعی برای سهولت در پیاده سازی الگوریتم نهایی تعریف کنیم.

۱ - تابع sign
این تابع با درواقع همان تابع np.sign است با این تفاوت که به‌جای 0، مقدار 1- را جاگذاری می‌کند.

```
def sign(a: np.ndarray):
    w = np.sign(a)
    w = w.astype(float)
    for i in range(len(w)):
        if w[i] == 0:
            w[i] = -1
    return w
```

۲ - تابع آسنکرون
این تابع با گرفتن بردارهای v_new و v_old، v_new را طوری تغییر می‌دهد که تنها یک تفاوت با v_old داشته باشد.

```
def asyncron(v_new, v_old):
    a = np.copy(v_old)
    for i in range(len(v_old)):
        if a[i] != v_new[i]:
            a[i] = v_new[i]
            break
    return a
```

۳ - تابع تبدیل تصویر به بردار

این تابع با گرفتن آدرس عکس مورد نظر، آن را خوانده و سپس با اعمال یک threshold به آن آن را بصورت یک بردار سطری در می‌آورد.

```
def image_info_extrackter(name):
    l1 = cv2.imread(name)
    l1 = cv2.resize(l1, (10, 10))
    l1 = cv2.cvtColor(l1, cv2.COLOR_BGR2GRAY)
    _, l1 = cv2.threshold(l1, 10, 255, 0)
    l1 = sign(np.reshape(l1, (1, -1))[0])
    return l1
```

۴ - تابع show image

این تابع با گرفتن بردار یک تصویر، آن را نمایش می‌دهد.

```
def show_image_from_info(input: np.ndarray):
    plt.imshow(input.reshape((10, 10)))
    plt.show()
```

۵ - تابع نویز

این تابع با گرفتن آدرس عکس مورد نظر، آن را خوانده و سپس با توجه با میزان درصد مشخص شده به آن نویز می دهد.

```
def noisy(name, percent=10):
  out = image_info_extrackter(name)
  l = np.random.randint(100, size=percent).tolist()
  for i in l:
    if out[i] == -1:
      out[i] = 1
    else:
      out[i] = -1
  return out
```

۵ - تابع debug

این تابع صرفا به منظور debug بوده و با گرفتن ۲ بردار، به منظور مقایسه آنها را کنار یکدیگر چاپ می‌کند.

```
def debug_print(a: np.ndarray, b: np.ndarray):
    for i in range(len(a)):
        print(a[i], b[i])
```

حال با استفاده از توابع بالا و الگوریتم داده شده و بصورت زیر عمل می‌کنیم.

ابتدا تصاویر داده شده (الگو ها) را خوانده و بردارشان را extract می‌کنیم.

```
photo_b = image_info_extrackter("photo_b.jpg")
photo_d = image_info_extrackter("photo_d.jpg")
photo_g = image_info_extrackter("photo_g.jpg")
photo_k = image_info_extrackter("photo_k.jpg")
photo_p = image_info_extrackter("photo_p.jpg")
```

سپس ورودی‌ها را بوسیله‌ی دادن نویز به همان الگوها تعیین می‌کنیم.

```
noisy_b = noisy("photo_b.jpg", 5)
noisy_d = noisy("photo_d.jpg", 0)
noisy_g = noisy("photo_g.jpg", 5)
noisy_k = noisy("photo_k.jpg", 5)
noisy_p = noisy("photo_p.jpg", 5)
```

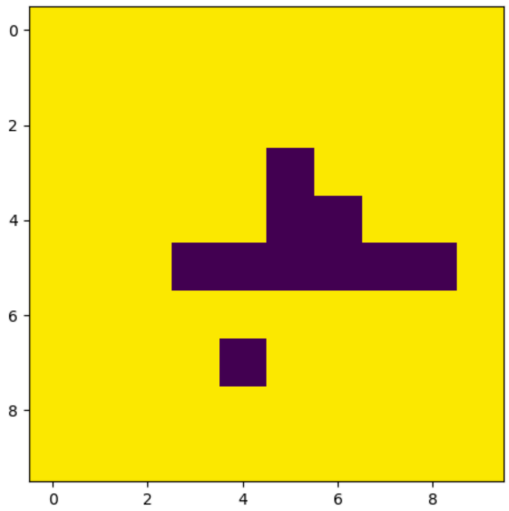
حال با توجه به الگوریتم گفته شده، ماتریس W را می‌سازیم.

```
pattern = np.transpose(np.concatenate(([photo_b], [photo_d], [photo_g], [photo_k], [photo_p]), axis=0))
m = pattern.shape[0]
n = pattern.shape[1]
W = np.dot(pattern, np.transpose(pattern)) - n * np.eye(m)
```

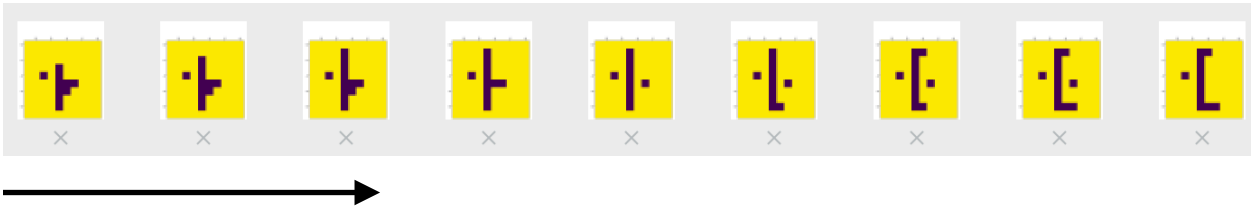
و در آخر با توجه به الگوریتم داده شده، ورودی را گرفته و پله پله خود را به نزدیکترین الگو می‌رسانیم.

```
input = photo_b
show_image_from_info(input)
v0 = np.transpose(np.array([input]))

v_old = v0
counter = 0
v_new = np.ones((len(v_old), 1))

while True:
    v_new_temp = sign(np.dot(W, v_old))
    counter += 1
    show_image_from_info(v_new)
    print("iter {} -> {}".format(counter, np.transpose(v_new)))
    if np.allclose(v_old, v_new):
        print("result -> {}".format(np.transpose(v_new)))
        show_image_from_info(v_new)
        break
    v_old = v_new
```

برای مثال با دادن تصویر 'ب' نویزی شده به صورت زیر الگوریتم را دنبال می‌کنیم.



الگوریتم پس از ۹ iteration به صورت زیر به تعادل می‌رسد.



گزارش و کد را می‌توانید در repository زیر ببینید(Hopefield)

https://github.com/kianbehzad/computational-intelligence