به نام خدا

دانشگاه تهران

دانشکده مهندسی برق و کامپیوتر

شبکه های عصبی و یادگیری عمیق

تمرین سری چهارم

نام و نام خانوادگی	علی عدالت
شماره دانشجویی	۸۱۰۱۹۹۳۴۸
تاریخ ارسال گزارش	

فهرست گزارش سوالات (لطفاً پس از تکمیل گزارش، این فهرست را به‌روز کنید.)

سوال 1 – SOM

در اینجا ابتدا از طریق keras مجموعه داده MNIST را لود می‌کنیم. سپس داده‌های آموزش را با توجه به لیبل به ۱۰ دسته تقسیم می‌کنیم و از هر دسته ۲۰۰ نمونه تصادفی برای ۲۰۰۰ داده آموزش انتخاب می‌کنیم. داده‌های تست از مجموعه داده اصلی mnist را نیز مانند قبل به ۱۰ دسته بر اساس لیبل تقسیم می‌کنیم و از هر دسته ۱۰۰ نمونه به تصادف بر می داریم تا ۱۰۰۰ داده تست را تشکیل دهیم. در Figure 1 این عملیات تشکیل مجموعه تست و آموزش از داده اصلی آمده است.

```
(X_train, y_train), (X_test, y_test) = mnist.load_data()
```

```
ids = [[] for i in range(10)]
for i in range(len(y_train)):
  ids[y_train[i]].append(i)
```

```
x_tr = []
y_tr = []
pis = [list(np.random.choice(ids[i], 200, replace=False)) for i in range(10)]

for i in pis:
  for j in i:
    x_tr.append(X_train[j])
    y_tr.append(y_train[j])
```

```
ids = [[] for i in range(10)]
for i in range(len(y_test)):
  ids[y_test[i]].append(i)
```

```
x_te = []
y_te = []
pis = [list(np.random.choice(ids[i], 100, replace=False)) for i in range(10)]

for i in pis:
  for j in i:
    x_te.append(X_test[j])
    y_te.append(y_test[j])
```

```
X_train = np.array(x_tr)
X_test = np.array(x_te)

y_train = np.array(y_tr)
y_test = np.array(y_te)
```

Figure 1 عملیات تشکیل مجموعه تست و آموزش از داده اصلی

بعد از تشکیل مجموعه‌های تست و آموزش، داده‌های هر مجموعه را shuffle می‌کنیم تا ترتیب قرار گیری در مجموعه‌ها در تحلیل بی اثر گردد. همچنین بازه‌ی اعداد پیکسل‌ها از ۰ تا ۲۵۵ است. برای نرمال سازی تصاویر، اعداد هر پیکسل تصویر را باید بین صفر و یک قرار دهیم تا مشکلی در محاسبات بوجود نیاید. این مورد در Figure 2 آمده است. در Figure 2 نحوه به هم ریختن داده‌های هر دسته آمده است. یک نمونه از داده‌های آموزش در Figure 3 آمده است.

```
from sklearn.utils import shuffle
X_train, y_train = shuffle(X_train, y_train, random_state=0)
X_test, y_test = shuffle(X_test, y_test, random_state=0)
```

```
# Change to float datatype
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')

# Noramlization from [0;255] to [0;1], Scale the data to lie between 0 to 1
X_train /= 255
X_test /= 255
```

Figure 2 در هم ریختن داده‌های هر مجموعه تست و آموزش و نرمالسازی داده‌ها

```
plt.imshow(X_train[600], cmap='gray'), y_train[600]
```

```
(<matplotlib.image.AxesImage at 0x7ff045eea6d0>, 3)
```

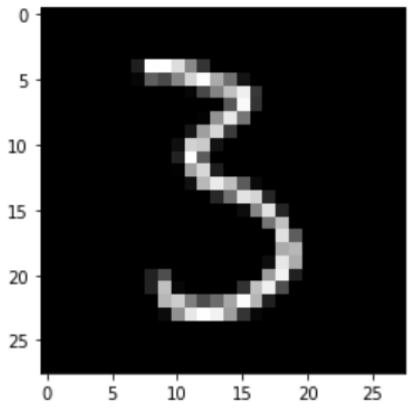

Figure 3 یک نمونه از داده‌های آموزش

الف

برای استفاده از الگوریتم SOM باید مقدار نرخ یادگیری اولیه، شعاع همسایگی اولیه، تعداد کلاسترهای مورد نظر و نحوه قرار گیری نقاط متناظر کلاسترها کنار هم تعیین شوند. در اینجا برای کلاستر کردن از ۶۲۵ تا کلاستر استفاده می‌کنیم. همسایگی نقاط به صورت خطی است. یعنی تمام نقاط متناظر کلاسترها روی یک خط بافاصله یکسان یک واحد از هم قرار دارند. در اینجا شعاع همسایگی صفر است. یعنی فقط کلاستر با کمترین فاصله از داده که برنده است، به روز می‌شود. دلیل آن همسایگی صفر است. یعنی در ابتدا فرآیند هم کاری وجود ندارد و از ابتدا نماینده‌های کلاسترها به رقابت می‌پردازند تا در تمرکزهای داده‌ها قرار گیرند. اتدا برای هر نماینده وزن‌های راندم در نظر می‌گیریم. به ازای هر داده نماینده‌ای که از همه به داده نزدیک تر است را به روز می‌کنیم. این به روز کردن به این شکل است که وزن‌های نماینده را به داده نزدیک می‌کنیم. اگر شعاع همسایگی صفر نباشد، تمام دیگر نماینده‌ها در شعاع همسایگی به مرکز نماینده برنده را هم، به روز می‌کنیم. این در نظر گرفتن همسایگی با توجه به شکل قرار گیری نماینده‌ها کنار هم است. بعد از دیدن تک تک داده ها یک iteration انجام شده است و بعد از آن باید اندازه شعاع همسایگی و نرخ یادگیری به روز شود. به تعداد مورد نیاز iteration انجام می‌دهیم تا کلاستریگ به خوبی انجام شود. یک معیار اتمام این است که آنقدر انجام می‌دهیم تا نرخ یادگیری به حدی برسد که دیگر عملا یادگیری نداشته باشیم. نحوه انجام الگوریتم و در نظر گرفتن همسایگی در Figure 4 آمده است.

```
def neighborhood(j, r, m, t='l'):
  if t == 'l':
    res = []
    for i in range(m):
      if i >= j-r and i <= j+r:
        res.append(i)
    return res
  else:
    res = []
    for i in range(m):
      if (i%25) >= (j%25)-r and (i%25) <= (j%25)+r and int(i/25) >= int(j/25)-r and int(i/25) <= int(j/25)+r:
        res.append(i)
    return res
```

```
def som(xs, t='l', m=625, r=0, a=0.6):
  ws = np.random.rand(len(xs[0].reshape(-1)),m)
  t = 0
  while t < 10:
    print('epoch ', t)
    print('a = ', a, ' r = ', r)
    for x in xs:
      itx = x.reshape(1,-1).T
      ds = np.sum((ws-itx)**2, axis=0)
      j = np.argmin(ds)
      js = neighborhood(j, r, m, t=t)
      for j in js:
        for i in range(ws.shape[0]):
          ws[i][j] += a*(itx[i][0]-ws[i][j])
    a *= 0.5
    r = int(r*0.5)
    t += 1
```

Figure 4 نحوه انجام الگوریتم و در نظر گرفتن همسایگی

در اینجا از نرخ یادگیری 0.6 شروع می‌کنیمو بعد از هر iteration نرخ یادگیری را نصف می‌کنیم. شعاع همسایگی همیشه صفر است. برای یادگیری از ۱۰ تا iteration استفاده می‌کنیم. بعد از یادگیری به کلاستریگ داده‌های تست می‌پردازیم. فاصله اقلیدسی هر داده تا نماینده‌ها را بدست می‌آوریم. داده به کلاستر نزدیک ترین نماینده تعلق دارد. نتیجه کلاستریگ داده تست بعد از یادگیری در این شرایط به صورت Figure 5 است.

```
def predict(xa, ya, ws):
  res = [[] for i in range(ws.shape[1])]
  for i, x in enumerate(xa):
      itx = x.reshape(1,-1).T
      ds = np.sum((ws-itx)**2, axis=0)
      j = np.argmin(ds)
      res[j].append(ya[i])

  return res
```

```
preds = predict(X_test, y_test, ws)
```

```
import pandas as pd

clusters = []
for i in preds:
  if len(i) > 0:
    clusters.append(i)

pd.DataFrame(clusters)
```

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	2
0	9	8	2	5	6	9	0	2	3	7	4	5	7	2	5	0	9	1	0	0	0	4

1 rows × 1000 columns

Figure 5 نتیجه کلاستریگ داده تست بعد از یادگیری در شرایط بخش الف سوال ۱

همانطور که دیده می‌شود، تمام داده‌ها در یک کلاستر قرار گرفته اند. همانطور که دیده می شود لیبل داده‌های درون کلاستر متفاوت و از تمام اعداد ممکن است. این نشان دهنده این است که کلاستریک مطلوب نیست. کلاستریگی مطلوب است که داده‌های درون هر کلاستر به هم نزدیک و داده‌های دو کلاستر از هم دور باشند. در اینجا داده‌های درون یک کلاستر به هم نزدیک نیست. دلیل این موضوع این است که ما با شعاع صفر، فاز هم کاری نداریم و در ابتدا به رقابت می پردازیم. به نظر می‌رسد کل داده‌ها در یک ناحیه تمرکز بزرگ قرار دارند که در این ناحیه زیر ناحیه‌های تمرکز کوچک تر قرار دارد. چون فاز هم کاری نداریم، یک نماینده به هر ناحیه تمرکز می‌فرستیم و آن را نماینده کل قرار می‌دهیم. این باعث شده که یک نماینده برای کل تمرکز داده‌ها داشته باشیم. این باعث تک کلاستر شدن یک تمرکز با تنوع و کلاستریگ بی کیفیت شده است.

ب

در اینجا مانند قبل عمل می‌کنیم ولی شعاع همسایگی ابتدایی ۲ داریم. بعد از هر iteration شعاع را نصف می‌کنیم و کف مقدار جدید را در نظر می‌گیریم. این باعث می‌شود در ابتدا همکاری و بعد از صفر شدن شعاع، رقابت داشته باشیم. نتایج با همسایگی خطی و شعاع دو در Figure 6 آمده است.

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	6	6.0	6.0	6.0	6.0	6.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1	6	6.0	6.0	6.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	4	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	0	0.0	0.0	0.0	0.0	0.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	0	0.0	0.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...	...
414	8	8.0	8.0	8.0	8.0	8.0	8.0	NaN	NaN	NaN	NaN	NaN	NaN
415	8	8.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
416	2	8.0	8.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
417	8	8.0	8.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
418	8	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

419 rows × 13 columns

Figure 6 نتایج با همسایگی خطی و شعاع دو

همانطور که دیده می‌شود، ۴۱۹ کلاستر برنده داریم. همانطور که دیده می‌شود لیبل داده‌های داخل کلاسترها اکثرا مشابه است. یعنی داده‌های درون کلاسترها به هم نزدیک و کلاسترها از هم دور اند. این نشان می دهد که کلاسترینگ خوبی داشته ایم. دلیل آن وجود مرحله هم کاری در ابتدا و فرستادن چند نماینده به یک تمرکز است. این باعث می‌شود زیر تمرکزهای یک تمرکز را بتوانیم کلاستر کنیم و این باعث ایجاد کلاستریگ مطلوب شده است. کل داده‌ها یک تمرکز را ساخته است. وقتی شعاع ابتدا صفر نیست ما تا زمان صفر شدن شعاع هم کاری داریم. این فاز هم کاری باعث می‌شود برای پوشش تمرکز در کل داده‌ها از چندین نماینده استفاده کنیم. وجود مرحله هم کاری در ابتدا باعث فرستادن چند نماینده به یک تمرکز است. این باعث می‌شود زیر تمرکزهای یک تمرکز را بتوانیم کلاستر کنیم و این باعث ایجاد کلاستریگ مطلوب شده است. به همین دلیل کلاستریگ داده‌های تست مطلوب انجام شده چون توانستیم برای زیر تمرکزها هم نماینده داشته باشیم. با پوشش مطلوب زیر تمرکزها نسبت به دفعه قبل، توانسته‌ایم برای داده‌های دیده نشده کلاسترینگ خوبی داشته باشیم. چون پوشش مناسب باعث قدرت تعمیم بالاتر کلاسترینگ می‌شود.

ج

در اینجا از شعاع یک استفاده می‌کنیم و نرخ کاهش شعاع و نرخ یادگیری را مانند قبل قرار می‌دهیم. نرخ یادگیری اولیه هم مانند قبل است. تفاوت این مرحله با قبل در مقدار شعاع اولیه و شکل قرار گیری و همسایگی نماینده‌ها است. در این جا بعد از بررسی هر داده، علاوه بر نماینده برنده نماینده‌های داخل مربع با شعاع یک اطراف آن را نیز به روز می‌کنیم. همسایگی مطابق Figure 7 است.

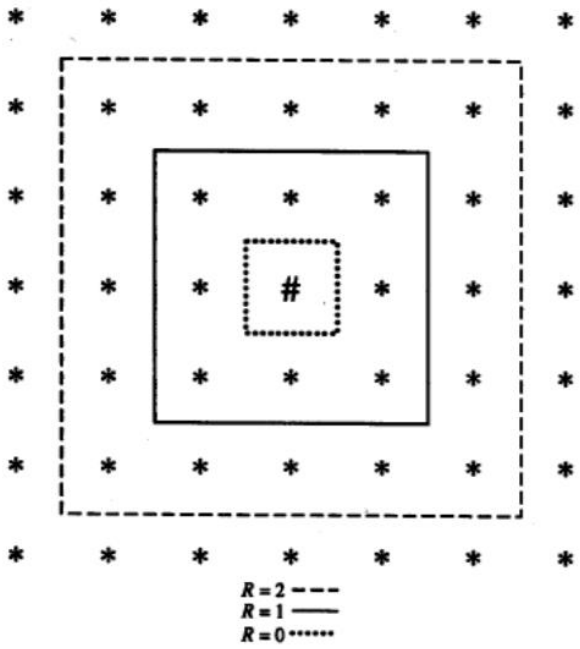

Figure 7 نماینده‌های داخل مربع با شعاع یک اطراف برنده

بعد از یادگیری بعد از ۱۰ iteration مقدار نرخ یادگیری بسیار پایین می‌آید و بعد آن یادگیری چشم گیری نخواهیم داشت. بعد از ۱۰ تا iteration یادگیری کامل شده است. بعد از یادگیری وزن‌ها متناظر تعدادی از نماینده‌ها را می‌خواهیم نشان دهیم. برای این کار به کلاستر کردن داده‌های تست می‌پردازیم و از میان نماینده‌های برنده ۳ تا را انتخاب می‌کنیم. در Figure 8 و Figure 9 و Figure 10 وزن متناظر سه نماینده انتخابی به صورت عکس آمده است. همچنین لیبل واقعی داده‌های نسبت داده شده به این نماینده‌ها نیز در عکس‌ها آمده است. همانطور که دیده می‌شود عکس نماینده‌ها به ارقام مجموعه داده نزدیک است. نکته دیگر این است که لیبل داده‌های نسبت داده شده به هر نماینده مشابهت بالایی به عکس نماینده دارند. یعنی داده‌ها درون خوشه نزدیک به هم و داده‌های خوشه‌ها از هم فاصله دارند. که نشان دهنده یک کلاستریگ خوب است.

```
(<matplotlib.image.AxesImage at 0x7ff03b00d7d0>, [2, 2, 2, 2])
```

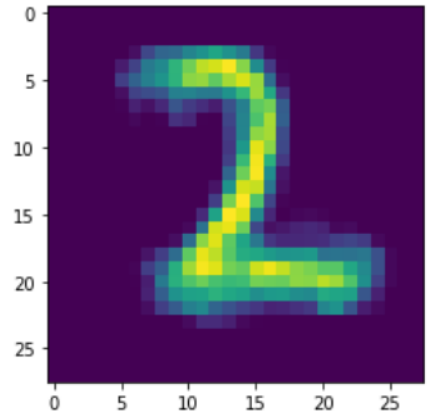

Figure 8 یک نماینده برنده بر روی داده تست به صورت عکس به همراه لیبل داده‌های نماینده، نماینده عدد ۲

```
(<matplotlib.image.AxesImage at 0x7ff03b0c23d0>, [4, 4, 4, 4, 4, 4])
```

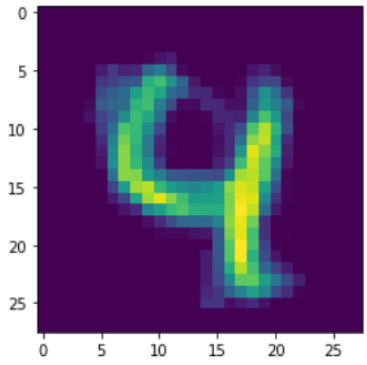

Figure 9 یک نماینده برنده بر روی داده تست به صورت عکس به همراه لیبل داده‌های نماینده، نماینده عدد ۴

```
(<matplotlib.image.AxesImage at 0x7ff03b10c390>, [9, 4, 4, 9, 4, 9])
```

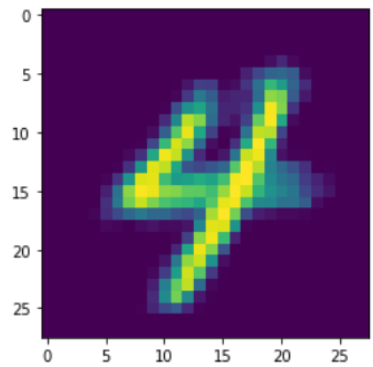

Figure 10 یک نماینده برنده بر روی داده تست به صورت عکس به همراه لیبل داده‌های نماینده، نماینده عدد ۴

سوال ۲ – MaxNet

در اینجا به پیاده سازی شبکه Maxnet می‌پردازیم. در این شبکه در هر epoch تمام داده‌ها را به روز می‌کنیم. برای به روز کردن هر داده به صورت Figure 11 عمل می‌کنیم. هر داده را منهای اپسیلون برابر مجموع دیگر داده‌ها می‌کنیم. حاصل از تابع فعالساز relu عبور می‌دهیم.

Step 0. Initialize activations and weights (set $0 < \epsilon < \frac{1}{m}$):

$a_j(0)$ input to node A_j,

$$w_{ij} = \begin{cases} 1 & \text{if } i = j; \\ -\epsilon & \text{if } i \neq j. \end{cases}$$

Step 1. While stopping condition is false, do Steps 2–4.

Step 2. Update the activation of each node: For $j = 1, \ldots, m$,

$$a_j(\text{new}) = f[a_j(\text{old}) - \epsilon \sum_{k \neq j} a_k(\text{old})].$$

Step 3. Save activations for use in next iteration:

$$a_j(\text{old}) = a_j(\text{new}), j = 1, \ldots, m.$$

Step 4. Test stopping condition:
If more than one node has a nonzero activation, continue; otherwise, stop.

Figure 11 به روز کردن هر داده

این به روز رسانی را آنقدر انجام می‌دهیم که فقط یک عدد غیر صفر متناظر داده‌ها اولیه داشته باشیم یا تعدادی عدد غیر صفر برابر متناظر اعداد اوب داشته باشیم. اگر فقط یک عدد غیر صفر داشته باشیم، مقدار بیشینه فقط یک بار بین اعداد آمده است. اگر چند عدد برابر غیر صفر داشته باشیم، نشان دهنده وجود چند بار تکرار مقدار بیشینه در بین اعداد است. در این شرایط نیز مقدار متناظر یکی از این اعداد غیر صفر بیشینه است. به همین دلیل بعد از این شرایط دیگر به بروز رسانی ادامه نمی‌دهیم. الگوریتم در Figure 12 آمده است.

```
def act(x):
  if x >= 0:
    return x
  else:
    return 0
```

```
def sum_j(x,i):
  return sum(x)-x[i]
```

```
def maxnet(x, e):
  print(x)
  xt = x.copy()
  while len(np.nonzero(xt)[0]) > 1:
    for i in range(len(x)):
      xt[i] = act(xt[i]-(e*sum_j(x,i)))
    print(xt)
  return np.nonzero(xt)[0][0], x[np.nonzero(xt)[0][0]]
```

Figure 12 الگوریتم پیاده سازی شده

عملکرد گام به گام الگوریتم برای پیدا کردن مقدار بیشینه آرایه در Figure 13 آمده است. مقدار اپسیلون مطابق مقدار پیشنهادی استفاده شده است.

```
i, v = maxnet(x, e)
print('max value: ', v, ' index of max : ', i, ' input : ', x)
```

```
[1.2, 1.1, 0.5, 1.5, 1.13, 0.8]
[0.5461, 0.43310000000000015, 0, 0.8851, 0.46699999999999997, 0.09410000000000007]
[0, 0, 0, 0.2702, 0, 0]
max value:  1.5  index of max :  3  input :  [1.2, 1.1, 0.5, 1.5, 1.13, 0.8]
```

Figure 13 عملکرد گام به گام الگوریتم برای پیدا کردن مقدار بیشینه آرایه

همانطور که دیده می‌شود، بعد از دو گام یا epoch فقط یک عدد غیر صفر داریم. Index آن برابر سه است. یعنی در index سه ارایه اصلی مقدار بیشینه وجود دارد که برابر 1.5 است. همانطور که دیده می‌شود به درستی مکان و مقدار بیشینه مقدار در آرایه بدست آمده است.

برای بدست آوردن مقدار بیشینه قدر مطلق اعداد آرایه، ابتدا دو شبکه maxnet تشکیل می‌دهیم. یک شبکه بیشینه اعداد آرایه اصلی را پیدا می‌کند و شبکه دیگر بیشینه 1- ضرب در المان‌های آرایه اصلی یا قرینه آرایه اصلی را پیدا می‌کند. بعد از اتمام کار این شبکه‌ها، یک آرایه شامل بیشینه خود ورودی و قرینه آن تشکیل می‌دهیم. به کمک یک شبکه maxnet دیگر بیشینه بین این دو مقدار را پیدا می‌کنیم. از اپسیلون پیشنهادی در بخش قبل استفاده می‌کنیم. در اینجا شبکه شامل دو maxnet روی خود ورودی و قرینه آن است. خروجی دو شبکه با هم کانکت می‌شوند و به یک شبکه maxnet دیگر داده می‌شود تا خروجی نهایی بدست آید. الگوریتم شبکه بیشینه قدر مطلق اعداد در Figure 14 آمده است. عملکرد گام به گام شبکه در Figure 15 به همراه خروجی نهایی آمده است. همانطور که دیده می‌شود، خروجی نهایی مطابق بیشینه قدرمطلق اعداد درون آرایه است.

```
def max_abs_net(x, e):
  y = []

  print('find max positive number')
  print('use max net on input array')
  print(x)
  xt = x.copy()
  while len(np.nonzero(xt)[0]) > 1:
    for i in range(len(xt)):
      xt[i] = act(xt[i]-(e*sum_j(xt,i)))
    print(xt)
  y.append(x[np.nonzero(xt)[0][0]])

  print('find min negative number')
  print('use max net on  -1 * input array')
  xt = x.copy()
  xt = list(np.array(xt)*-1)
  in_xt = xt.copy()
  print(xt)
  while len(np.nonzero(xt)[0]) > 1:
    for i in range(len(xt)):
      xt[i] = act(xt[i]-(e*sum_j(xt,i)))
    print(xt)
  y.append(in_xt[np.nonzero(xt)[0][0]])
  print(y)
  print('find max betwwen abs of min negative and max positive as max absolute')
  xt = y.copy()
  while len(np.nonzero(xt)[0]) > 1:
    for i in range(len(xt)):
      xt[i] = act(xt[i]-(e*sum_j(xt,i)))
    print(xt)
  return y[np.nonzero(xt)[0][0]]
```

Figure 14 الگوریتم شبکه بیشینه قدر مطلق اعداد

```
max_abs_net(x, e)
```

```
find max positive number
use max net on input array
[1.2, 1.1, 0.5, -1.5, 1.13, -0.8]
[1.1441, 1.038367, 0.36837929, 0, 0.9023899822999999, 0]
[0.8439122846009999, 0.76345839760287, 0.04211040361449697, 0, 0.6879574411436122, 0]
[0.6497538730940726, 0.5840815742820864, 0, 0, 0.5275588329847116, 0]
[0.5052406201493889, 0.4498176453746533, 0, 0, 0.40340125846658614, 0]
[0.39432216265002773, 0.3461136006294935, 0, 0, 0.3071446092402484, 0]
[0.3093985953669613, 0.26596298403055624, 0, 0, 0.2323476039185712, 0]
[0.24461821893357472, 0.20395742705977726, 0, 0, 0.17403276993943537, 0]
[0.19547949332367706, 0.15592083283557265, 0, 0, 0.12835072753873292, 0]
[0.15852419047501734, 0.11862709349378511, 0, 0, 0.09232106062278858, 0]
[0.13110093043986276, 0.08958223465564044, 0, 0, 0.06363224916037316, 0]
[0.111183047543781, 0.066856246084l004, 0, 0, 0.040487140988748574, 0]
[0.09722840722431063, 0.048953224816402706, 0, 0, 0.02148352882345584, 0]
[0.08807162925112902, 0.034711054266706676, 0, 0, 0.005521779966137199, 0]
[0.08284136080085931, 0.023223845966997128, 0, 0, 0, 0]
[0.07982226082514969, 0.012846952059727667, 0, 0, 0, 0]
[0.0781521570573851, 0.002687171642267604, 0, 0, 0, 0]
[0.0778028247438903, 0, 0, 0, 0, 0]
find min negative number
use max net on  -1 * input array
[-1.2, -1.1, -0.5, 1.5, -1.13, 0.8]
[0, 0, 0, 1.5429, 0, 0.599423]
[0, 0, 0, 1.4649750099999999, 0, 0.40897624870000004]
[0, 0, 0, 1.4118080976689997, 0, 0.2254411960030301]
[0, 0, 0, 1.3825007421886057, 0, 0.04571609951851133]
[0, 0, 0, 1.3765576492511993, 0, 0]
[1.2, 1.5]
find max betwwen abs of min negative and max positive as max absolute
[1.005, 1.36935]
[0.8269844999999999, 1.261842015]
[0.6629450380499999, 1.1756591600535]
[0.5101093472430449, 1.1093449449119042]
[0.36589450440449733, 1.0617786593393195]
[0.2278632786903858, 1.0321564331095694]
[0.09368294238614178, 1.019977650599371]
[0, 1.019977650599371]
```

```
1.5
```

Figure 15 عملکرد گام به گام شبکه به همراه خروجی نهایی

سوال 3 – Mexican Hat

در اینجا به پیاده سازی این شبکه و آموزش در دو حالت گفته شده می‌پردازیم. در اینجا از تابع فعالساز به صورت Figure 16 استفاده می‌کنیم.

$$f(x) = \begin{cases} 0 & \text{if } x < 0 \\ x & \text{if } 0 \le x \le 2 \\ 2 & \text{if } 2 < x. \end{cases}$$

Figure 16 تابع فعالساز

در این شبکه می‌خواهیم، مقدار بیشینه نرم در آرایه را تعیین کنیم. بیشینه نرم ناحیه‌ای از آرایه است که در آن مقدار بیشینه قرار دارد. مقدار بیشنه و همسایه‌های آن تا شعاعی از آن در این ناحیه است. با توجه به این موضوع، تا همسایه‌ها به شعاع R1 اطراف هر مقدار به بیشینه شدن آن کمک می‌کنند و همسایه‌ها از شعاع R1 تا شعاع R2 اطراف آن با بیشینه شدن رقابت می‌کنند. سعی می کنند مقدار مرکز بیشینه نشود. بقیه مقادیر خارج این بازه ها هیچ تاثیری ندارند. در حالت اول شعاع هم کاری را صفر و شعاع رقابت را بی نهایت قرار می‌دهیم. یعنی در بیشینه شدن هر مقدار فقط خود آن عدد ماثر است و عدد مورد نظر با تمام دیگر اعداد برای بیشینه شدن مبارزه می‌کند. در این الگوریتم در هر iteration ، تمام مقادیر را به روز می کنیم. برای به روز کردن هر مقدار از جمع تمام اعداد داخل شعاع همکاری با وزن w1 و جمع تمام اعداد داخل ناحیه رقابت با وزن w2 به صورت Figure 17 استفاده می‌کنیم.

$$x_i = C_1 \sum_{k=-R_1}^{R_1} x\_old_{i+k} + C_2 \sum_{k=-R_2}^{-R_1-1} x\_old_{i+k} + C_2 \sum_{k=R_1+1}^{R_2} x\_old_{i+k}.$$

Figure 17 نحوه به روز کردن هر مقدار

در اینجا مقادیر ci همان وزن‌های w1 و w2 هستند. در اینجا از وزن 0.5 برای همکاری و از وزن منهای 0.4 برای رقابت استفاده می کنیم. در طی یادگیری و بعد از هر iteration آرایه مقادیر به روز شده و نمودار مقادیر را خروجی می‌دهیم. در اینجا وزن‌ها تعیین میکنند که چقدر همکارها و رقبا در تعیین مقدار جدید تاثیر بگذارند. یعنی چقدر همکارها و رقبا در تعیین ناحیه بیشینه نرم تاثیر دارند. در اینجا برای پیدا کردن بیشینه نرم در دو حالت، از ۳ تا iteration استفاده می‌کنیم. الگوریتم در Figure 18 آمده است. خروجی گام به گام الگوریتم در حالت اول در Figure 19 تا Figure 24 آمده است.

```
def mexican_hat(x,r1, r2, w1=0.6, w2=0.4):
  xt = x.copy()
  it = 0
  while it < 3:
    for i in range(len(xt)):
      res = 0
      for j in range(len(xt)):
        res += get_w(i,j,r1,r2,w1,w2)*xt[j]
      xt[i] = res
    xt = list(map(act, xt))
    print(xt)
    plt.plot(xt)
    plt.xlabel('xi')
    plt.ylabel('x')
    plt.grid()
    plt.show()
    it += 1
```

```
r1, r2 = 0, float('inf')
mexican_hat(X,r1, r2, w1=0.5, w2=0.4)
```

Figure 18 الگوریتم کلاه مکزیکی

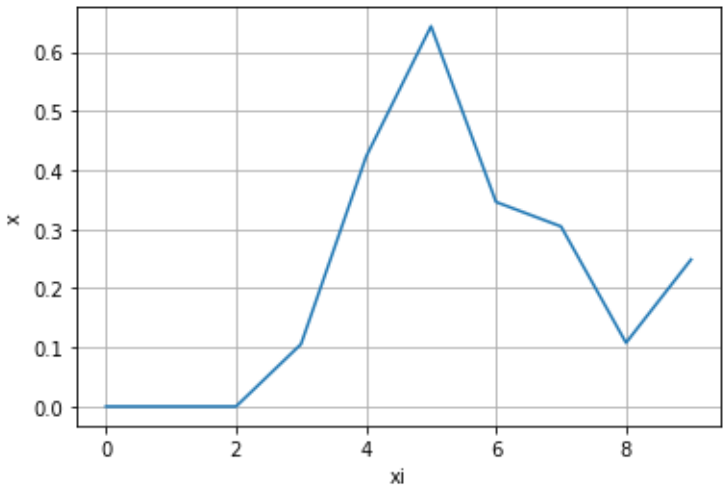

Figure 19 گام اول در حالت اول

```
[0, 0, 0,
0.10544799999999988,
0.4222688, 0.64336128,
0.34601676800000014,
0.3046100608000001,
0.10776603648000002,
0.24865962188800003]
```

Figure 20 خروجی سیگنال گام اول حالت اول

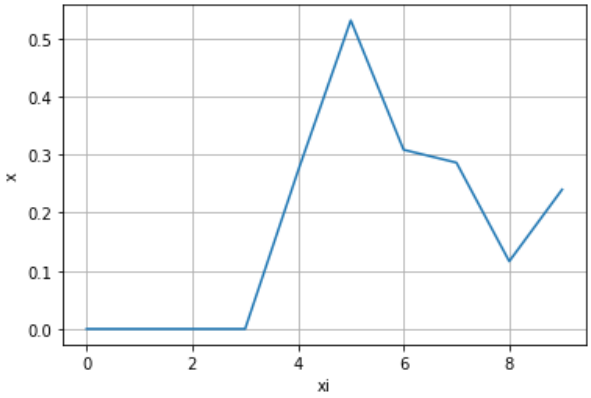

Figure 21 گام دوم حالت اول

```
[0, 0, 0, 0,
0.27134555139801103,
0.5306980828388066,
0.308153300903284,
0.28603265126197036,
0.11630399318918223,
0.23969303737270936]
```

Figure 22 خروجی گام دوم حالت اول

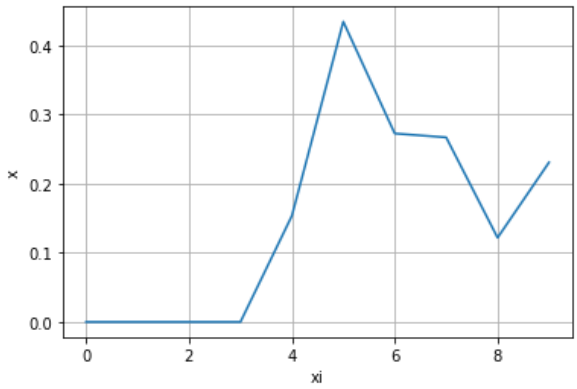

Figure 23 گام سوم حالت اول

```
[0, 0, 0, 0,
0.15337556843479805,
0.43398083991679914,
0.2723774333436319,
0.26677919569031056,
0.12172478565346515,
0.2306066084329264]
```

Figure 24 خروجی گام سوم حالت اول

همانطور که در خروجی حالت اول دیده می‌شود، قله کلاه فقط بر روی مقدار بیشینه است و بقیه مقادیر فاصله زیادی از قله دارند. در اینجا قله تعیین کننده‌ی بیشینه مقدار در میان اعداد است. خروجی گام به گام الگوریتم در حالت دوم در Figure 25 تا Figure 28 آمده است.

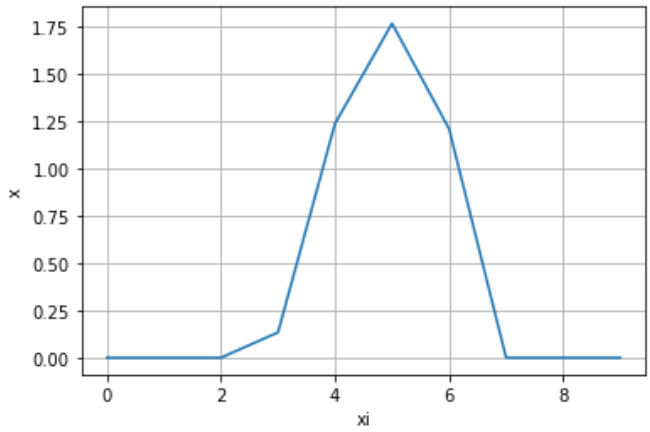

Figure 25 گام اول حالت دوم

```
[0, 0, 0,
0.13387499999999988,
1.2393975,
1.7686087499999998,
1.210855375, 0, 0, 0]
```

Figure 26 خروجی گام اول حالت دوم

```
[0, 0, 0, 0, 2, 2, 2, 0, 0, 0]
```

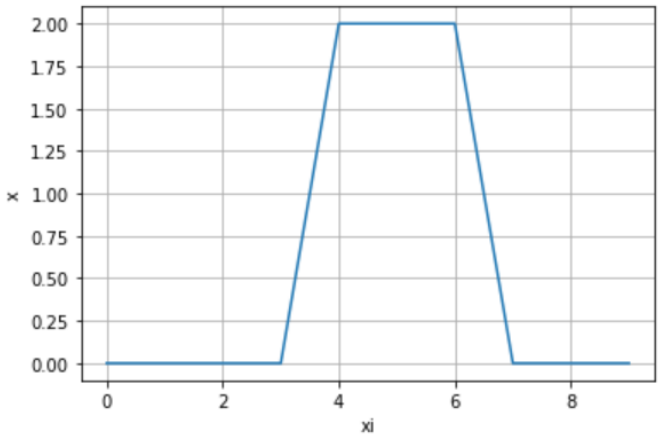

Figure 27 خروجی گام دوم حالت دوم

```
[0, 0, 0, 0, 2, 2, 2, 0, 0, 0]
```

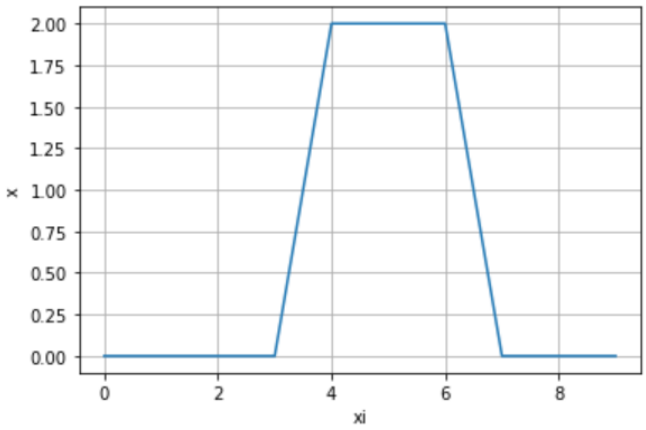

Figure 28 خروجی گام سوم حالت دوم

همانطور که دیده می‌شود، در این حالت قله کلاه بر روی سه مقدار 0.4 ,0.80, 0.66 قرار دارد. این قله شامل همسایگی از مقدار بیشینه است که اعداد درون همسایگی مقادیر بزرگی هستند. در اینجا مقدار شعاع همسایگی یک است. یعنی به دنبال سه عدد پشت هم هستیم که از اعداد با مقادیر بالا در آرایه باشند و در ۴ عدد قبل و بعد از سه عدد اعداد کوچکتری با فاصله نسبتا زیادی از سه عدد باشد. ناحیه سه عدد قله کلاه است. این ناحیه تعیین شده بر روی آرایه اعداد اولیه کاندید پاسخ برای بیشینه نرم است. پاسخ با انتظار تطابق دارد.

سوال 4 – Hamming Net

الف

در اینجا تعدادی بردار پایه داریم. می خواهیم بردار ورودی را به نزدیک ترین بردار پایه نسبت بدهیم. فاصله دو بردار را به صورت فاصله همینگ تعریف می‌کنیم. فاصله همینگ دو بردار برابر بعد بردار منهای مشابهت دو بردار است. پس به جای پیدا کردن بردار پایه با کمترین فاصله از ورودی می‌توان بردار پایه با بیشترین مشابهت به ورودی را پیدا کرد. رابطه فاصله و مشابهت و رابطه ضرب دو بردار با این موارد در Figure 29 آمده است.

For bipolar vectors *x* and y with n dimension.

$$x \cdot y = a - d,$$

a= number of equal bits

d= number of not equal bits

Also, it is known

$$d = n - a$$

Figure 29 رابطه فاصله و مشابهت و رابطه ضرب دو بردار با این موارد

بر اساس تعاریف Figure 29 می‌توان میزان مشابهت بردار ورودی و یک بردار پایه را به صورت Figure 30 تعریف کرد. در این رابطه $\frac{y}{2}$ ماتریس وزن‌ها و $\frac{n}{2}$ بایاس است.

$$a = x\left(\frac{y}{2}\right) + \left(\frac{n}{2}\right)$$

Figure 30 میزان مشابهت بردار ورودی و یک بردار پایه

پس بر اساس این رابطه می‌توان شبکه‌ای تعریف کرد که مقدار مشابهت بردار ورودی x را با بردار پایه y محاسبه کند. این شبکه در Figure 31 آمده است.

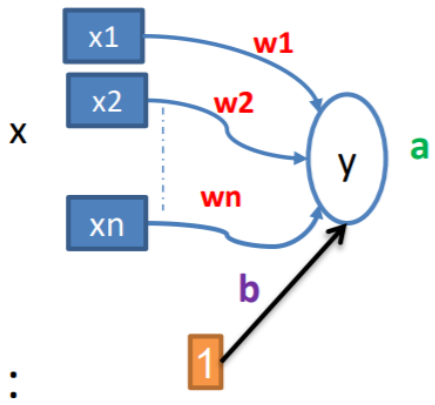

Figure 31 شبکه محاسبه مشابهت ورودی با بردار پایه **y**

برای پیدا کردن بردار پایه با بیشترین مشابهت کافی است به ازای هر بردار پایه y_i شبکه‌ای مانند بالا ایجاد کنیم و به تمام این شبکه‌ها بردار ورودی را بدهیم. این شبکه‌ها به صورت موازی مشابهت ورودی با هر یک از بردارهای پایه را محاسبه می‌کنند. برای پیدا کردن بردار پایه مورد نظر باید y_i با بیشترین مشابهت را پیدا کنیم که برای این کار از شبکه MaxNet استفاده می‌کنیم. ساختار شبکه hamming برای پیدا کردن مشابه ترین بردار پایه به ورودی در Figure 32 آمده است.

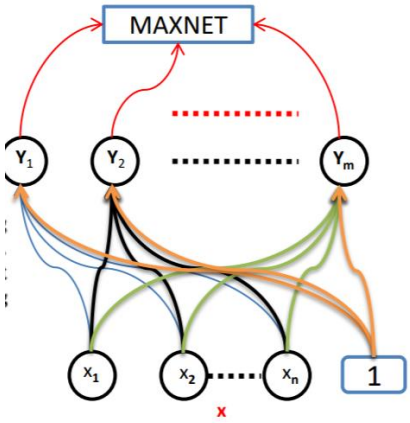

Figure 32 ساختار شبکه **hamming** برای پیدا کردن مشابه ترین بردار پایه به ورودی

ب

برای پیاده سازی مانند Figure 32 عمل می‌کنیم. پیاده سازی شبکه MaxNet در Figure 33 آمده است.

```
def multimax(x):
  t = x[np.nonzero(x)]
  return np.all(t==t[0])
```

```
def act(x):
  if x >= 0:
    return x
  else:
    return 0
```

```
def sum_j(x,i):
  return sum(x)-x[i]
```

```
def maxnet(x, e):
  print(x)
  xt = x.copy()
  while not multimax(xt) and len(np.nonzero(xt)[0]) > 1:
    for i in range(len(x)):
      xt[i] = act(xt[i]-(e*sum_j(x,i)))
    print(xt)
  return np.nonzero(xt)[0][0], x[np.nonzero(xt)[0][0]]
```

Figure 33 پیاده سازی شبکه **MaxNet**

با استفاده از maxNet پیاده سازی شده و ساختار شبکه گفته شده در الف شبکه را به صورت Figure 34 پیاده سازی می‌کنیم. در maxnet از اپسیلون 0.13 استفاده می‌کنیم.

```
def hamming_net(es, v, ep):
  xt = np.matmul(v, np.array(es).T)+(len(v)/2)
  i, v1 = maxnet(xt, ep)
  print('base vec of ', v, ' is ', 'e', i, ' = ', es[i])
```

Figure 34 پیاده سازی شبکه همینگ

بعد از پیاده سازی شبکه، تک تک بردارهای ورودی را به شبکه می‌دهیم تا نزدیک ترین بردار پایه به آنها را بیابد. خروجی شبکه برای بردارهای v_i به ترتیب در Figure 35 تا Figure 36 آمده است.

```
hamming_net(es, V1, ep)
```

```
[3. 1. 3.]
[2.48 0.22 2.48]
[1.96 0.   1.96]
base vec of  [1, 1, 1, 1, 1, 1]  is  e 0  =  [1, -1, 1, -1, 1, -1]
```

```
hamming_net(es, V2, ep)
```

```
[1. 3. 1.]
[0.48 2.74 0.48]
[0.   2.48 0.  ]
base vec of  [-1, 1, -1, -1, 1, 1]  is  e 1  =  [-1, 1, -1, 1, -1, -1]
```

Figure 35 خروجی شبکه همینگ برای دو بردار یک و دو ورودی

```
hamming_net(es, V3, ep)
```

```
[ 3.  1. -1.]
[3.   0.74 0.  ]
[3.   0.48 0.  ]
[3.   0.22 0.  ]
[3. 0. 0.]
base vec of  [-1, -1, 1, 1, 1, 1]  is  e 0  =  [1, -1, 1, -1, 1, -1]
```

```
hamming_net(es, V4, ep)
```

```
[1. 3. 1.]
[0.48 2.74 0.48]
[0.   2.48 0.  ]
base vec of  [-1, -1, 1, 1, -1, 1]  is  e 1  =  [-1, 1, -1, 1, -1, -1]
```

```
hamming_net(es, V5, ep)
```

```
[3. 5. 7.]
[1.44 3.7  5.96]
[0.   2.4  4.92]
[0.   1.1  3.88]
[0.   0.   2.84]
base vec of  [-1, 1, 1, -1, -1, -1]  is  e 2  =  [1, 1, 1, -1, -1, -1]
```

Figure 36 خروجی شبکه همینگ برای ورودی‌های سه تا ۵