به نام خدا

دانشگاه تهران

دانشکده مهندسی برق و کامپیوتر

شبکه های عصبی و یادگیری عمیق

تمرین سری سه

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 – شناسایی حروف با استفاده از روش هب

الف

بله. شبکه توانایی این را دارد که برای تمام ورودی‌های بدون نویز و بدون حذف اطلاعات، خروجی متناظر را پیدا کند. در این روش شبکه یک ماتریس وزن است که بر اساس قاعده هب باید در طی آموزش بدست آید. در Figure 1 روش بدست آوردن ماتریس وزن به روش هب آمده است.

```
def gen_w(ins, outs):
  res = np.zeros((63, 15))
  for i, v in enumerate(ins):
    res += np.matmul(v.flatten()[np.newaxis].T ,outs[i].flatten().reshape(1, -1))
  return res
```

Figure 1 روش بدست آوردن ماتریس وزن به روش هب

برای بدست آوردن این ماتریس باید تمام ورودی‌ها و خروجی‌ها را به شکل ماتریس bipolar در بیاوریم و به تابع تولید ماتریس w بدهیم. یک نمونه از تبدیل ورودی به ماتریس در Figure 2 آمده است. یک نمونه از تبدیل خروجی به ماتریس در Figure 3 آمده است.

```
ins0 = [[-1,-1,-1,1,-1,-1,-1], [-1,-1,-1,1,-1,-1,-1], [-1,-1,-1,1,-1,-1,-1],
        [-1,-1,1,-1,1,-1,-1], [-1,-1,1,-1,1,-1,-1], [-1,1,1,1,1,1,-1],
        [-1,1,-1,-1,-1,1,-1], [-1,1,-1,-1,-1,1,-1], [1,1,1,-1,1,1,1]]

plt.imshow(np.array(ins0))
```

```
<matplotlib.image.AxesImage at 0x7f64769f1f10>
```

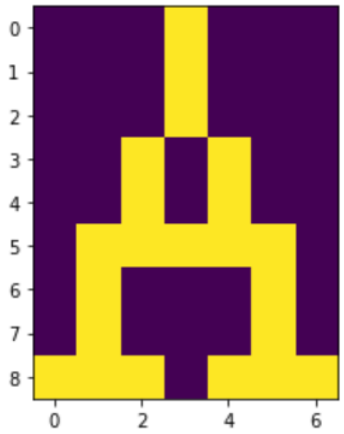


Figure 2 یک نمونه از تبدیل ورودی به ماتریس

```
outs0 = [[-1,1,-1], [1,-1,1], [1,1,1], [1,-1,1], [1,-1,1]]

plt.imshow(np.array(outs0))
```

```
<matplotlib.image.AxesImage at 0x7f64763e85d0>
```

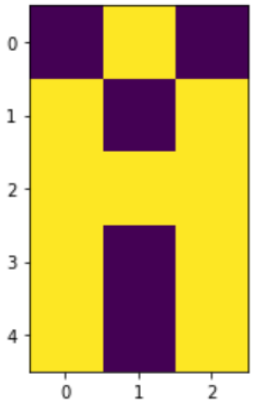
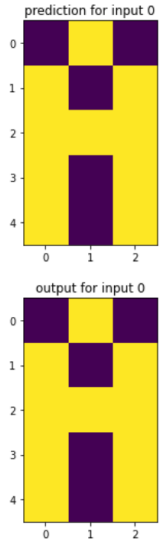


Figure 3 یک نمونه از تبدیل خروجی به ماتریس

بعد از تشکیل ماتریس وزن‌ها یا w برای بررسی توانایی شبکه برای تداعی خروجی درست برای ورودی‌های بدون تغییر، هر ورودی بدون تغییر را به شبکه می‌دهیم و خروجی شبکه برای آن را با خروجی واقعی مقایسه می‌کنیم. در Figure 4 خروجی پیش بینی شده شبکه برای ورودی اول که A است، مورد بررسی قرار گرفته است.




```
[[ True  True  True]
 [ True  True  True]
 [ True  True  True]
 [ True  True  True]
 [ True  True  True]]
True
```

Figure 4 خروجی پیش بینی شده شبکه برای ورودی اول که **A** است

در Figure 5 خروجی پیش بینی شده شبکه برای ورودی دوم که B است، مورد بررسی قرار گرفته است.

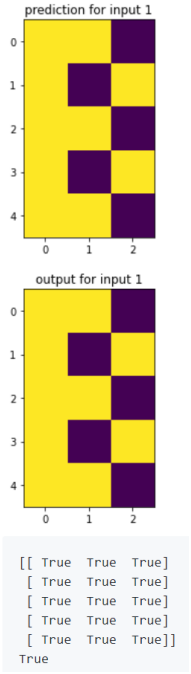


Figure 5 خروجی پیش بینی شده شبکه برای ورودی دوم که **B** است

در Figure 6 خروجی پیش بینی شده شبکه برای ورودی سوم که C است، مورد بررسی قرار گرفته است.

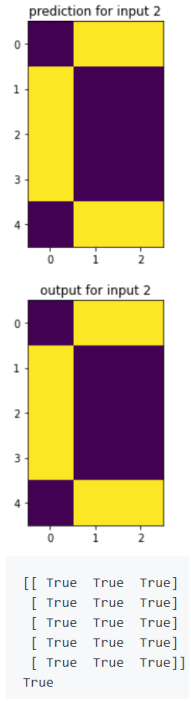


Figure 6 خروجی پیش بینی شده شبکه برای ورودی سوم که **C** است

همانطور که در Figure 4 و Figure 5 و Figure 6 دیده می‌شود، شبکه به ازای هر ورودی بدون تغییر خروجی درست را تداعی می‌کند.

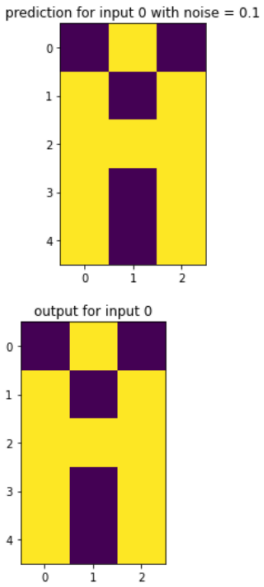
ب

در اینجا به هر ورودی یکبار ۱۰ درصد و بار دیگر ۲۵ درصد نویز اضافه می‌کنیم. پیش بینی شبکه برای هر ورودی با نویز را با خروجی درست مقایسه می‌کنیم. برای اضافه کردن نویز به صورت Figure 7 عمل می‌کنیم.

```
def add_noise(inp, p):
  ids = np.random.choice(len(inp), int(len(inp)*p)+1)
  inp_c = inp.copy()
  for i in ids:
    inp_c[i] += float((-2*inp_c[i])/np.abs(inp_c[i]))
  return inp_c
```

Figure 7 اضافه کردن نویز

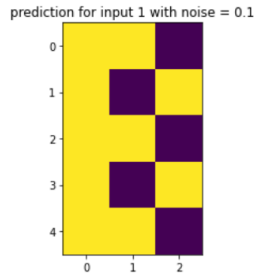
در ادامه هر حالت نویز برای هر ورودی را با خروجی درست مقایسه می‌کنیم. در Figure 8 بررسی پیش بینی برای ورودی A با نویز 10 درصد آمده است. همانطور که دیده می‌شود، پیش بینی کاملا با خروجی درست مطابقت دارد.

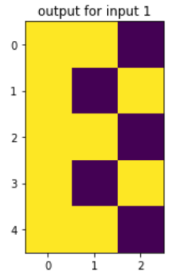



```
[[ True  True  True]
 [ True  True  True]
 [ True  True  True]
 [ True  True  True]
 [ True  True  True]]
True
```

Figure 8 پیش بینی برای ورودی **A** با نویز 10 درصد

در Figure 9 بررسی پیش بینی برای ورودی B با نویز 10 درصد آمده است. همانطور که دیده می‌شود، پیش بینی کاملا با خروجی درست مطابقت دارد.

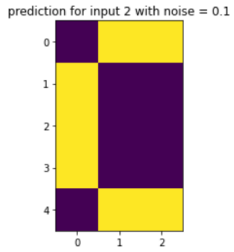


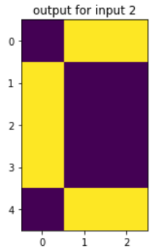



```
[[ True  True  True]
 [ True  True  True]
 [ True  True  True]
 [ True  True  True]
 [ True  True  True]]
True
```

Figure 9 پیش بینی برای ورودی **B** با نویز 10 درصد

در Figure 10 بررسی پیش بینی برای ورودی C با نویز 10 درصد آمده است. همانطور که دیده می‌شود، پیش بینی کاملا با خروجی درست مطابقت دارد.






```
[[ True  True  True]
 [ True  True  True]
 [ True  True  True]
 [ True  True  True]
 [ True  True  True]]
True
```

Figure 10 پیش بینی برای ورودی **C** با نویز 10 درصد

در Figure 11 بررسی پیش بینی برای ورودی A با نویز ۲۵ درصد آمده است. همانطور که دیده می‌شود، پیش بینی کاملا با خروجی درست مطابقت دارد.

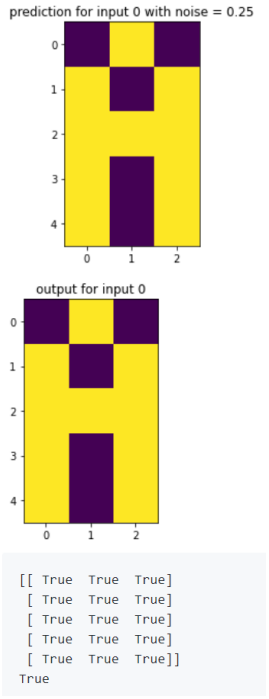


Figure 11 پیش بینی برای ورودی **A** با نویز ۲۵ درصد

در Figure 12 بررسی پیش بینی برای ورودی B با نویز ۲۵ درصد آمده است. همانطور که دیده می‌شود، پیش بینی با خروجی درست مطابقت دارد.

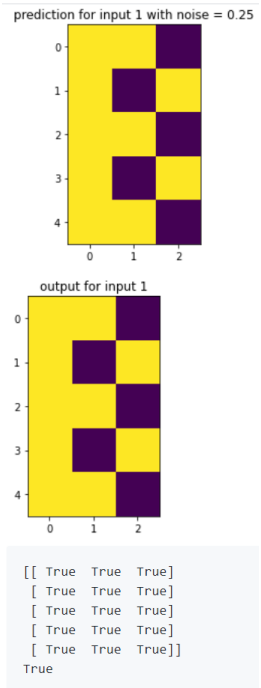
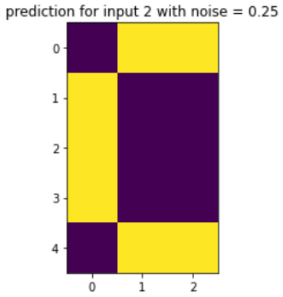
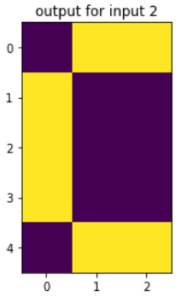


Figure 12 پیش بینی برای ورودی **B** با نویز ۲۵ درصد

در Figure 13 بررسی پیش بینی برای ورودی C با نویز ۲۵ درصد آمده است. همانطور که دیده می‌شود، پیش بینی کاملا با خروجی درست مطابقت دارد.






```
[[ True  True  True]
 [ True  True  True]
 [ True  True  True]
 [ True  True  True]
 [ True  True  True]]
True
```

Figure 13 پیش بینی برای ورودی **C** با نویز ۲۵ درصد

در اینجا درصد موفقیت برای تمام حالات ورودی و نویز را بدست می‌آوریم. از آنجا که ما به صورت راندم درایه‌های دارای نویز را تعیین می‌کنیم، پس ممکن است یک بار موفق شویم و در بار دیگر اجرا، نویز به گونه‌ای باشد که موفق به تداعی خروجی درست نشویم. به همین دلیل برای بدست آوردن درصد موفقیت مدل برای هر حالت ورودی نویز دار، باید چندین بار ورودی مورد نظر را به صورت راندم نویز دهیم و درصد باری از این اجرا ها که موفق به تداعی درست بوده‌ایم را اعلام کنیم. برای این کار برای هر ورودی و هر درصد نویز از ۱۰۰۰ بار اجرا استفاده می‌کنیم. در Figure 14 درصد موفقیت برای حالات مختلف ورودی‌های نویز دار آمده است.

```
{'input': 0, 'p_noise': 0.1, 'num of correct': 1000, 'rate of correct': 1.0}
{'input': 1, 'p_noise': 0.1, 'num of correct': 1000, 'rate of correct': 1.0}
{'input': 2, 'p_noise': 0.1, 'num of correct': 999, 'rate of correct': 0.999}
{'input': 0, 'p_noise': 0.25, 'num of correct': 997, 'rate of correct': 0.997}
{'input': 1, 'p_noise': 0.25, 'num of correct': 967, 'rate of correct': 0.967}
{'input': 2, 'p_noise': 0.25, 'num of correct': 968, 'rate of correct': 0.968}
```

Figure 14 درصد موفقیت برای حالات مختلف ورودی‌های نویز دار

همانطور که در Figure 14 دیده می‌شود، در ۱۰ درصد نویز برای تمام ورودی‌ها درصد موفقیت بسیار به صد نزدیک است. یعنی شبکه می‌تواند به خوبی با ۱۰ درصد نویز در داده ورودی، خروجی مناسب را تداعی کند. یعنی شبکه استحکام خوبی در برابر نویز ۱۰ درصد دارد. در حالت C درصد موفقیت کمتر از دیگر حروف با ۱۰ درصد نویز است. این یعنی این مقدار نویز باعث می‌شود تداعی درست این حرف دشوارتر شود. در ۲۵ درصد نویز می‌بینیم که درصدهای موفقیت از صد فاصله بیشتری دارند. در ورودی c با ۲۵ درصد نویز، ما 3.2 درصد موارد موفق به تداعی خروجی درست نمی‌شویم. در ورودی B نیز 3.3 درصد تداعی درستی نداریم. این یعنی این مقدار نویز تداعی این دو حرف را دشوارتر از حروف دیگر می‌کند. با افزایش درصد نویز، تعداد درایه بیشتری داده اشتباه دارد. دیدیم که با افزایش نویز درصد موفقیت برای تداعی حروف کمتر می‌شود. در دو حالت نویز حرف A درصد موفقیتی نزدیک به صد دارد. یعنی ساختار این ورودی به شکلی است که با توجه به ورودی‌های دیگر می‌تواند استحکام خوبی در برابر نویز داشته باشد.

در اینجا برای هر ورودی یکبار ۱۰ درصد و بار دیگر ۲۵ درصد درایه‌ها را به صورت راندم مانند قبل صفر می‌کنیم. در ادامه هر حالت صفر کردن برای هر ورودی را با خروجی درست مقایسه می‌کنیم. در Figure 15 بررسی پیش بینی برای ورودی A با 10 درصد صفر کردن آمده است. همانطور که دیده می‌شود، پیش بینی کاملا با خروجی درست مطابقت دارد.

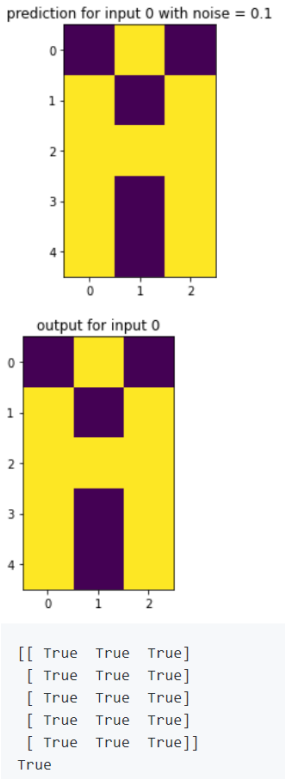


Figure 15 پیش بینی برای ورودی **A** با 10 درصد صفر کردن

در Figure 16 بررسی پیش بینی برای ورودی B با 10 درصد صفر کردن آمده است. همانطور که دیده می‌شود، پیش بینی کاملا با خروجی درست مطابقت دارد.

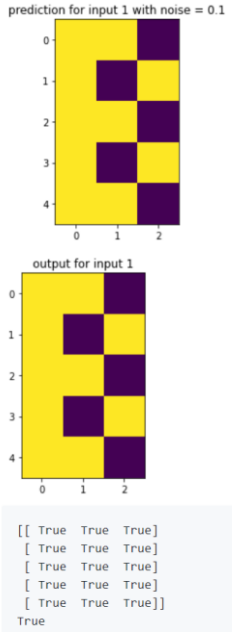


Figure 16 پیش بینی برای ورودی **B** با 10 درصد صفر کردن

در Figure 17 بررسی پیش بینی برای ورودی C با 10 درصد صفر کردن آمده است. همانطور که دیده می‌شود، پیش بینی کاملا با خروجی درست مطابقت دارد.

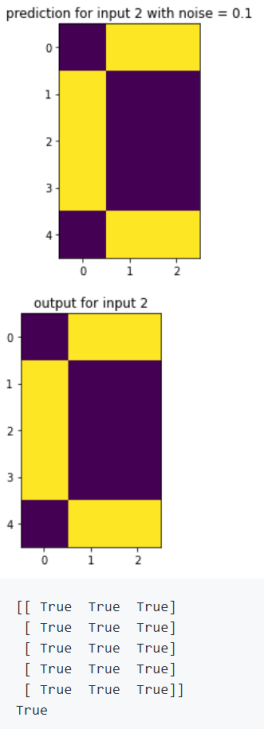
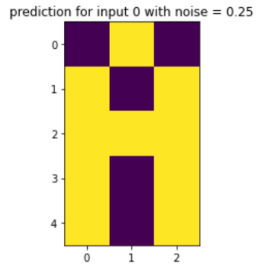
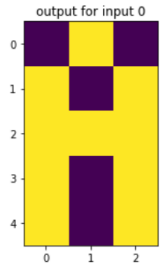


Figure 17 پیش بینی برای ورودی **C** با 10 درصد صفر کردن

در Figure 18 بررسی پیش بینی برای ورودی A با 25 درصد صفر کردن آمده است. همانطور که دیده می‌شود، پیش بینی کاملا با خروجی درست مطابقت دارد.

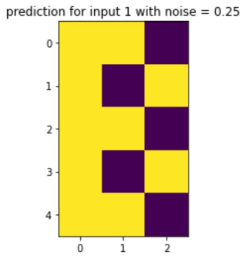


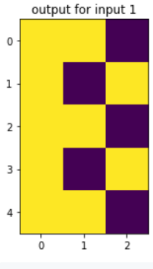



```
[[ True  True  True]
 [ True  True  True]
 [ True  True  True]
 [ True  True  True]
 [ True  True  True]]
True
```

Figure 18 پیش بینی برای ورودی **A** با 25 درصد صفر کردن

در Figure 19 بررسی پیش بینی برای ورودی B با 25 درصد صفر کردن آمده است. همانطور که دیده می‌شود، پیش بینی کاملا با خروجی درست مطابقت دارد.






```
[[ True  True  True]
 [ True  True  True]
 [ True  True  True]
 [ True  True  True]
 [ True  True  True]]
True
```

Figure 19 پیش بینی برای ورودی **B** با 25 درصد صفر کردن

در Figure 20 بررسی پیش بینی برای ورودی C با 25 درصد صفر کردن آمده است. همانطور که دیده می‌شود، پیش بینی کاملا با خروجی درست مطابقت دارد.

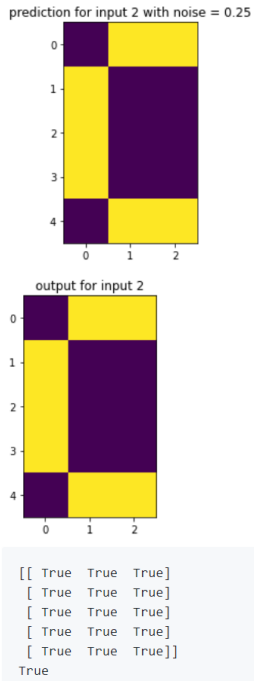


Figure 20 پیش بینی برای ورودی **C** با 25 درصد صفر کردن

مانند قبل برای بدست آوردن درصد موفقیت برای هر حالت صفر کردن برای هر ورودی عمل می‌کنیم. از ۱۰۰۰ بار اجرا برای بدست آوردن درصد موفقیت استفاده می‌کنیم. در Figure 21 درصد موفقیت برای حالت‌های مختلف صفرکردن ورودی‌های مختلف آمده است.

```
{'input': 0, 'p_remove': 0.1, 'num of correct': 1000, 'rate of correct': 1.0}
{'input': 1, 'p_remove': 0.1, 'num of correct': 1000, 'rate of correct': 1.0}
{'input': 2, 'p_remove': 0.1, 'num of correct': 1000, 'rate of correct': 1.0}
{'input': 0, 'p_remove': 0.25, 'num of correct': 1000, 'rate of correct': 1.0}
{'input': 1, 'p_remove': 0.25, 'num of correct': 1000, 'rate of correct': 1.0}
{'input': 2, 'p_remove': 0.25, 'num of correct': 1000, 'rate of correct': 1.0}
```

Figure 21 درصد موفقیت برای حالت‌های مختلف صفرکردن ورودی‌های مختلف

همانطور که دیده می‌شود، در تمام حالات درصد موفقیت صد درصد است. این یعنی شبکه در مقابل ۱۰ و ۲۵ درصد صفرکردن درایه‌ها استحکام خیلی خوبی دارد و در این حالات می‌تواند با صفر شدن درایه‌های ورودی، خروجی درست را در تمام مواقع تداعی کند.

سوال ۲ – شبکه خودانجمنی

الف

برای پیاده سازی این شبکه ابتدا باید به روش Modified Hebbian ماتریس وزن شبکه را تولید کنیم. بعد از تهیه ماتریس وزن‌ها، برای پیش بینی باید بردار ورودی را در ماتریس وزن‌ها ضرب کنیم و خروجی این ضرب را از تابع علامت عبور دهیم. بعد از انجام این فرآیند، پیش بینی شبکه برای ورودی یا تداعی بر اساس ورودی را داریم. در Figure 22 نحوه محاسبه ماتریس وزن‌ها به روش Modified Hebbian آمده است. ابتدا مانند روش هب ماتریس را می‌سازیم و بعد قطر آن را صفر می‌کنیم.

```
def gen_w(ins, outs):
  res = np.zeros((100, 100))
  for i, v in enumerate(ins):
    res += np.matmul(v.flatten()[np.newaxis].T ,outs[i].flatten().reshape(1, -1))
  return res
```

```
w = gen_w(ins, ins)

np.fill_diagonal(w, 0)
```

Figure 22 نحوه محاسبه ماتریس وزن‌ها به روش **Modified Hebbian**

در این جا به ازای هر مقدار R باید به اندازه‌ی R تا بردار ورودی راندم ایجاد کنیم. روش ساخت بردارهای ورودی برای هر مقدار R در Figure 23 آمده است.

```
ins = [(2*np.random.randint(2, size=N)-1).reshape(1,-1) for i in range(R)]
ins = np.array(ins)

w = gen_w(ins, ins)
```

Figure 23 روش ساخت بردارهای ورودی برای هر مقدار **R**

برای بررسی دقت شبکه، باید به هر ورودی نویز اضافه کنیم و به شبکه بدهیم تا شبکه پیش بینی بر اساس آن انجام دهد. E درصد نویز است. بر اساس این درصد تعدادی درایه از بردار ورودی را تغییر می‌دهیم. در Figure 7 روش اضافه کردن نویز آمده است. برای بررسی عملکرد از معیار مقایسه گفته شده برای خروجی شبکه و ورودی واقعی استفاده می‌کنیم. هر بار که ورودی نویز دار را به شبکه می‌دهیم، درصد درایه‌های مشابه خروجی شبکه و ورودی واقعی را بدست می‌آوریم. این درصد مشابهت را دقت می‌گوییم. متوسط دقت‌ها برای تمام ورودی‌های با یک مقدار مشخص نویز راندم را دقت اجرا با آن مقدار نویز E و تعداد ورودی R می‌نامیم. از آنجا که ما نویز را راندم اضافه می‌کنیم، ممکن است در هر بار اضافه کردن

نویز راندم به یک ورودی شبکه عملکرد متفاوتی داشته باشد. برای این که عملکرد واقعی شبکه به ازای یک مقدار نویز و تعداد مشخصی ورودی بدست بیاید، باید چندین اجرا انجام دهیم و متوسط دقت اجراها را برای عملکرد شبکه به ازای تعداد ورودی R و نویز E اعلام کنیم. در اینجا از ۳۰ اجرا استفاده می‌کنیم. در Figure 24 نمودار متوسط دقت به ازای هر تعداد ورودی R و هر نویز E آمده است.

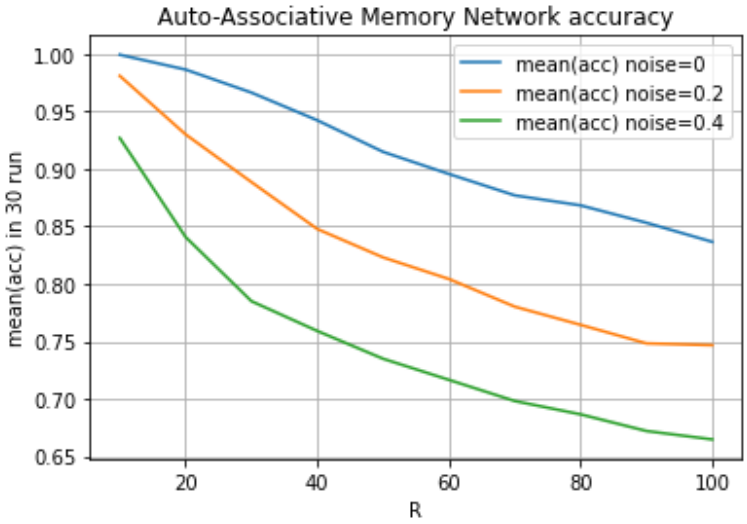


Figure 24 نمودار متوسط دقت به ازای هر تعداد ورودی **R** و هر نویز **E**

ب

در این نوع شبکه هر بردار ورودی ذخیره شده، یک بردار ویژه از ماتریس وزن‌ها است. با افزایش تعداد ورودی‌ها با توجه به نحوه ایجاد بردارهای ورودی، احتمال متعامد بودن دو بردار ورودی کمتر می‌شود. این باعث می‌شود که تعداد بردارهای کمتری را بتوان در ماتریس وزن‌ها ذخیره کرد. این مورد کاهش تداعی درست برای بردارها را حتی بدون نویز در پی دارد. به همین دلیل می‌بینیم که روند نمودارها همگی کاهشی است. در انتها در بدترین حالت نمودارها به دقت ۵۰ درصد میل می‌کنند که به به معنی پاسخ شانسی است. به همین دلیل می‌بینیم که شیب کاهش نمودارها در آخر کمتر شده است. این موضوع در نمودار متوسط دقت برای حالت نویز صفر به وضوح مشاهده می‌شود. در این حالت بدون هیچ نویز و تعییر ورودی، با افزایش تعداد بردارهای اصلی دقت کاهش می‌یابد.

نکته دیگر این است که در یک R مشخص همیشه با افزایش درصد نویز مقدار دقت کاهش می‌یابد. دلیل این موضوع این است که با افزایش نویز درایه‌های بیشتری اشتباه هستند و تداعی بردار اصلی سخت تر می‌شود. اگر نویز ورودی در فضای پوچی ماتریس وزن‌های شبکه تعریف شده باشد، شبکه در برابر نویز

مقاومت می‌کند و می‌تواند تداعی درستی داشته باشد. با افزایش نویز ورودی، قرار گیری تمام نویز در فضای پوچی احتمال کمتری پیدا می‌کند. یعنی مقداری از نویز خارج این فضا است و شبکه را دچار اشتباه می‌کند. به همین دلیل است که هر چه تعداد خانه دارای نویز بیشتر باشد، دقت کمتر خواهد بود. فاصله نمودارها به خاطر همین مقدار نویز است.

ج

در این شبکه در صورت عمود بودن ورودی‌های N بعدی با توجه به روش Modified Hebbian برای تولید ماتریس وزن، ظرفیت برابر N-1 خواهد بود. چون در این روش تولید ماتریس وزن، قطر اصلی را صفر کردیم. ظرفیت تعداد برداری است که می‌توان به خاطر سپرد. ظرفیت شبکه به رابطه بردارهای ورودی وابسته است. اگر بردارها متعامد باشند، تعداد بردار بیشتری را می‌توان به خاطر سپرد اما در نهایت تعداد بردارهای N بعدی مانند آنچه در ابتدا گفتیم برابر N-1 خواهد بود.

سوال 3 – شبکه هاپفیلد

الف

در این جا برای بدست آوردن ماتریس وزن‌ها از روش Modified Hebbian استفاده می‌کنیم. این روش در Figure 22 آمده است. بعد از تشکیل ماتریس ضرایب، برای تداعی ورودی اصلی از ورودی نویز دار مانند Figure 25 عمل می‌کنیم.

```
def hop_net(x, w, ins):
  rng = np.random.default_rng()
  y = x.copy()
  r = 0
  while r < 400:
    pr = rng.permutation(len(x[0]))
    for i in pr:
      y_in = x[0][i] + sumw(y, w, i)
      y[0][i] = act(y_in, y[0][i])

    # plt.imshow(y.reshape(8, 8))
    # print(y.reshape(8, 8))
    if convergence(y.reshape(8, 8), ins):
      return y
    r += 1

  print('conv false')
  print(x)
  for i, v in enumerate(ins):
    print('input : ', i, ' x ham-d : ', np.sum(x.reshape(8, 8) != v))
  plt.imshow(x.reshape(8,8))
  plt.show()
  return y
```

Figure 25 تداعی ورودی اصلی از ورودی نویز دار در شبکه هاپفیلد

در این جا ابتدا بردار y را برابر ورودی x قرار می‌دهیم. در هر epoch بر اساس ماتریس وزن‌ها و ورودی x باید یکی یکی درایه‌های y را به روز کنیم. برای این کار ترتیب راندمی از index درایه‌ها برای تعیین ترتیب به روز رسانی انتخاب می‌کنیم. برای به روز کردن هر درایه از y مانند i امین درایه، مجموع درایه i ام از x و درایه i ام از ضرب خارجی y و ماتریس ضرایب را بدست می‌آوریم. این مجموع را از تابع فعالساز عبور می‌دهیم تا درایه i ام از y به روز شود. برای به روز کردن دیگر درایه‌ها از y به روز شده در این مرحله استفاده می‌کنیم. تابع فعالساز و نحوه بدست آوردن درایه i ام از ضرب خارجی y و ماتریس ضرایب در Figure 26 آمده است.

```
def act(c, i):
  if c > 0:
    return 1
  elif c == 0:
    return i
  else:
    return -1
```

```
def sumw(y, w, i):
  res = 0
  for j in range(len(w)):
    res += y[0][j]*w[j][i]
  return res
```

Figure 26 تابع فعالساز و نحوه بدست آوردن درایه **i** ام از ضرب خارجی **y** و ماتریس ضرایب

بعد از یک epoch و به روز کردن تک تک درایه‌های y شرط همگرایی را چک می‌کنیم. اگر y بدست آمده برداری از بردارهای ورودی اصلی باشد، همگرا شده ایم. اگر این طور نبود، باید به انجام epoch بیشتر ادامه دهیم تا همگرا شویم یا به حداکثر تعداد epoch که ۴۰۰ تا است برسیم. البته اگر بعد از یک epoch مقدار y تغییری نکند نیز می‌توان کار را متوقف کرد. بر اساس ماتریس وزن‌ها و روش تداعی که در بالا گفتیم، شبکه بدست می‌آید. در ادامه بررسی می‌کنیم که شبکه توانایی تداعی ورودی بدون نویز را دارد یا نه. برای این کار تک تک ورودی ها را به شبکه می‌دهیم و خروجی را با ورودی مقایسه می‌کنیم. پیش بینی شبکه برای کلمه صفر بدون نویز در Figure 27 آمده است.

```
pred = hop_net(ins[0].reshape(1,-1), w, ins)
plt.imshow(pred.reshape(8, 8))
```

```
<matplotlib.image.AxesImage at 0x7f97d2e92650>
```

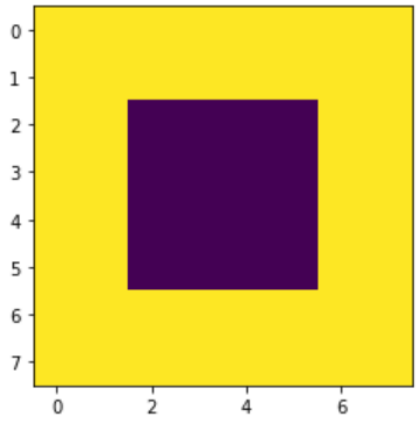


Figure 27 پیش بینی شبکه برای کلمه صفر بدون نویز

پیش بینی شبکه برای کلمه یک بدون نویز در Figure 28 آمده است. همانطور که دیده می‌شود، شبکه در دو حالت توانسته تداعی درستی داشته باشد.

```
pred = hop_net(ins[1].reshape(1,-1), w, ins)
plt.imshow(pred.reshape(8, 8))
```

```
<matplotlib.image.AxesImage at 0x7f97d2df7990>
```

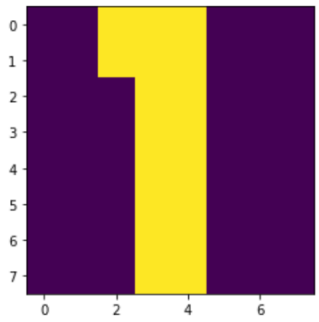
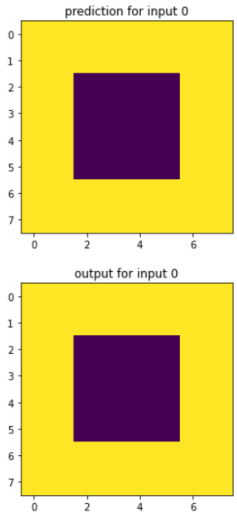


Figure 28 پیش بینی شبکه برای کلمه یک بدون نویز

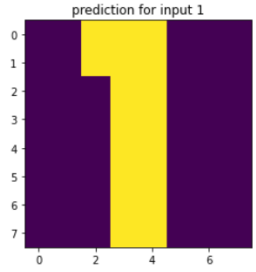
در بالا دیدیم که شبکه توانایی تداعی ورودی اصلی روی ورودی اصلی را دارد و عملا توانسته ورودی‌های اصلی را ذخیره کند. برای بررسی این که ماتریس وزن‌ها توانایی ذخیره ورودی‌ها را دارند، باید هر ورودی را در ماتریس وزن‌ها ضرب کنیم و خروجی ضرب را با ورودی مقایسه کنیم. در Figure 29 بررسی توانایی وزن شبکه برای ذخیره صفر آمده است.




```
[[ True  True  True  True  True  True  True  True]
 [ True  True  True  True  True  True  True  True]
 [ True  True  True  True  True  True  True  True]
 [ True  True  True  True  True  True  True  True]
 [ True  True  True  True  True  True  True  True]
 [ True  True  True  True  True  True  True  True]
 [ True  True  True  True  True  True  True  True]
 [ True  True  True  True  True  True  True  True]]
True
```

Figure 29 بررسی توانایی وزن شبکه برای ذخیره صفر

در Figure 30 بررسی توانایی وزن شبکه برای ذخیره یک آمده است. همانطور که دیدیم، ماتریس وزن‌ها توانایی ذخیره ورودی‌ها را دارد.




```
[[ True  True  True  True  True  True  True  True]
 [ True  True  True  True  True  True  True  True]
 [ True  True  True  True  True  True  True  True]
 [ True  True  True  True  True  True  True  True]
 [ True  True  True  True  True  True  True  True]
 [ True  True  True  True  True  True  True  True]
 [ True  True  True  True  True  True  True  True]
 [ True  True  True  True  True  True  True  True]]
True
```

Figure 30 بررسی توانایی وزن شبکه برای ذخیره یک

ب

در اینجا به هر عدد ۳۰ درصد نویز اضافه می‌کنیم و ورودی نویز دار را به شبکه می‌دهیم. خروجی شبکه را با ورودی اصلی مقایسه می‌کنیم تا عملکرد شبکه در برابر این میزان نویز روی ورودی ها را ببینیم. برای اضافه کردن نویز مانند Figure 7 عمل می‌کنیم. پیش بینی و تداعی شبکه برای صفر با نویز در Figure 31 آمده است. پیش بینی و تداعی شبکه برای یک با نویز در Figure 32 آمده است. همان طور که دیده می‌شود، شبکه برای دو ورودی با نویز توانسته ورودی‌های اصلی را تداعی کند. این مورد نشان می‌دهد که شبکه توانایی مقابله با این مقدار نویز را دارد. برای اینکه با آزمایش تصادفی بودن یا نبودن این مقابله صد درصد را بررسی کنیم، باید چندین بار به ورودی‌ها نویز دهیم و تداعی شبکه را با ورودی اصلی مقایسه کنیم. باید بررسی کنیم که از چندین بار دادن نویز به یک ورودی، چند بار شبکه می‌تواند برای آن ورودی تداعی درستی داشته باشد. دلیل انجام چندی بار این فرآیند بررسی عملکرد شبکه برای یک ورودی با نویز

این است که ما به صورت تصادفی به ورودی نویز اضافه می‌کنیم. به همین دلیل ممکن است مشاهده ما به دلیل آرایش خاص درایه‌های دارای نویز باشد و با تغییر این آرایش شبکه نتواند تداعی درستی داشته باشد.

```
pred = hop_net(add_noise(ins[0].reshape(1,-1), 0.3), w, ins)
plt.imshow(pred.reshape(8, 8))
```

```
<matplotlib.image.AxesImage at 0x7f97d2ddde10>
```

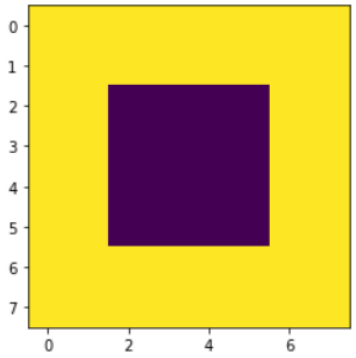


Figure 31 تداعی شبکه برای صفر با نویز

```
pred = hop_net(add_noise(ins[1].reshape(1,-1), 0.3), w, ins)
plt.imshow(pred.reshape(8, 8))
```

```
<matplotlib.image.AxesImage at 0x7f97d2d52210>
```

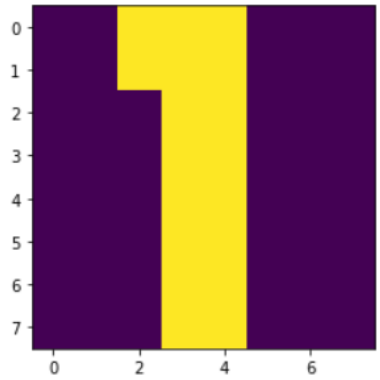


Figure 32 تداعی شبکه برای یک با نویز

در اینجا برای هر ورودی ۴۰۰ بار اضافه کردن نویز راندم را انجام می‌دهیم و نسبت دفعاتی که شبکه برای آن ورودی با نویز توانسته تداعی درستی داشته باشد را گزارش می‌کنیم. در Figure 33 عملیات بررسی عملکرد شبکه در برابر نویز آمده است.

```
accs = []
sh = 0
for i in range(len(ins)):
  acc = 0
  for r in range(400):
    pred = hop_net(add_noise(ins[i].reshape(1,-1), 0.2), w, ins)
    if not np.all(pred.reshape(8,8) == ins[i]):
      sh = (pred)
      plt.imshow(sh.reshape(8,8))
      plt.show()
      print('input : ', i)
      for i, v in enumerate(ins):
        print('input : ', i, ' y ham-d : ', np.sum(sh.reshape(8, 8) != v))
    acc += int(np.all(pred.reshape(8,8) == ins[i]))/400
  accs.append({'input':i, 'acc': round(acc, ndigits=4)})

for i in accs:
  print(i)
```

```
{'input': 0, 'acc': 1.0}
{'input': 1, 'acc': 1.0}
```

Figure 33 عملیات بررسی عملکرد شبکه در برابر نویز

همانطور که در Figure 33 دیده می‌شود، در تمام موارد اضافه کردن نویز به هر یک از ورودی‌ها، شبکه توانسته تداعی کاملا درستی را داشته باشد. این نشان می‌دهد که شبکه برای این دو ورودی می تواند در برابر ۳۰ درصد نویز مقاوم باشد و تداعی درستی از ورودی اصلی داشته باشد.

ج

در Figure 34 دو ورودی در کنار هم آمده است. همان طور که دیده می‌شود، دو ورودی در ۶ خانه بالای ورودی یک و چهار خانه پایین عدد یک که رنگ شده با هم اشتراک دارند. یعنی از ۶۴ خانه ورودی یک باید تمام خانه‌ها به جز این ۱۰ خانه از یک به منفی یک و بر عکس تغییر کنند تا به ورودی عدد صفر برسیم. برای رسیدن از صفر به یک نیز باید ۵۴ خانه تغییر کند.

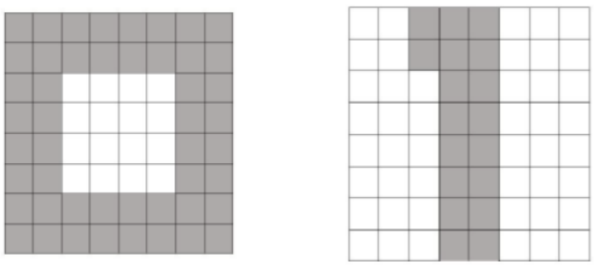


Figure 34 دو ورودی در کنار هم

این یعنی فاصله‌ی همینگ این دو ورودی ۵۴ واحد است. در اینجا ما از نویز ۳۰ درصد استفاده می‌کنیم. با توجه به این که ۶۴ خانه در هر ورودی داریم، در نویز ۳۰ درصد ما ۲۰ خانه را تغییر می‌دهیم. اگر ورودی یک باشد و این ۲۰ خانه را از خانه‌های غیر مشترک تغییر دهیم، ما ۲۰ واحد به ورودی صفر نزدیک می‌شویم. یعنی حاصل این نویز روی ورودی یک، فاصله ۲۰ از ورودی اصلی یک و ۳۴ از ورودی صفر دارد. اگر ورود یک باشد و تعدادی از ۲۰ خانه‌ای که تغییر می‌دهیم با ۱۰ خانه اشتراکی دو ورودی مشترک باشد، در این صورت ما به تعداد خانه‌های مشترک تغییر داده شده از صفر دور شده ایم و به اندازه ۲۰ منهای تعداد خانه‌های اشتراکی تغییر کرده به صفر نزدیک شده ایم. این یعنی خروجی حاصل از این نویز کمتر از ۲۰ واحد به صفر نزدیک است و فاصله بیش از ۳۴ واحد از صفر دارد. فاصله آن از یک همان ۲۰ است. این یعنی هر آرایشی از نویز ۳۰ درصد که به ورودی یک وارد کنیم، حاصل از نظر فاصله همینگ به یک نزدیک تر است و از صفر دور تر است. از آنجا که این شبکه سعی می‌کند ورودی را به ورودی اصلی نزدیکتر به آن برساند، پس به همین دلیل برای تمام حالات نویز برای ورودی یک، تداعی شبکه درست برابر ورودی یک اصلی است. برای ورودی صفر نیز به همین شکل می‌توان نشان داد که ورودی صفر با نویز به ورودی صفر اصلی نزدیک تر و از ورودی اصلی یک دورتر است. این یعنی در هر حالت نویز که به ورودی صفر اضافه کنیم، تداعی شبکه به درستی برابر ورودی اصلی صفر است. این یعنی شبکه در تمام حالات نویز می‌تواند تداعی درستی داشته باشد. این موضوع با نتیجه دیده شده در قسمت (ب) مطابقت دارد.

سوال 4 – شبکه BAM

الف

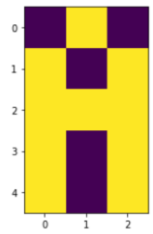
مانند Figure 1 به روش هب ماتریس وزن‌ها را برای سه حرف A و B و C تشکیل می‌دهیم. در Figure 35 ماتریس وزن‌های ساخته شده برای این حروف آمده است.

```
[[ 1. -1.  3.]
 [-3. -1. -1.]
 [ 1.  3. -1.]
 [-3. -1. -1.]
 [ 3.  1.  1.]
 [-1. -3.  1.]
 [-3. -1. -1.]
 [-1. -3.  1.]
 [ 1. -1. -1.]
 [-3. -1. -1.]
 [ 3.  1.  1.]
 [-1. -3.  1.]
 [-1. -3.  1.]
 [-1.  1.  1.]
 [-1.  1. -3.]]
```

Figure 35 ماتریس وزن‌ها برای سه حرف **A** و **B** و **C**

ب

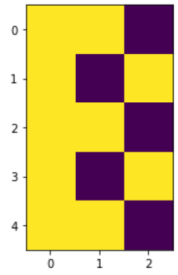
برای بررسی توانایی شبکه برای بازیابی اطلاعات از ورودی به خروجی و بالعکس، باید از ماتریس وزن‌های بدست آمده استفاده کنیم. باید بررسی کنیم که تمام ورودی‌ها با خروجی‌های متناظرشان در این ماتریس وزن ذخیره شده است یا نه. برای این کار ماتریس هر کلمه را در ماتریس وزن‌ها ضرب می‌کنیم و خروجی را با خروجی مورد انتظار با توجه به ورودی مقایسه می‌کنیم. برای بررسی خروجی‌ها، خروجی‌ها را در ترنسپوز ماتریس وزن‌ها ضرب می‌کنیم و حاصل را با ورودی متناظر مقایسه می‌کنیم. در Figure 36 کلمه A به عنوان ورودی داده شده و خروجی با بردار خروجی متناظر این کلمه مقایسه شده است. همانطور که دیده می‌شود، خروجی با خروجی متناظر ورودی مطابقت دارد.




```
[ True  True  True]
True
```

Figure 36 کلمه **A** به عنوان ورودی داده شده و خروجی با بردار خروجی متناظر این کلمه مقایسه شده

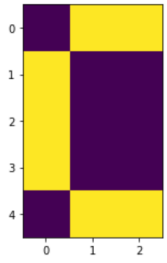
در Figure 37 کلمه B به عنوان ورودی داده شده و خروجی با بردار خروجی متناظر این کلمه مقایسه شده است. همانطور که دیده می‌شود، خروجی با خروجی متناظر ورودی مطابقت دارد.



```
[ True  True  True]
True
```

Figure 37 کلمه **B** به عنوان ورودی داده شده و خروجی با بردار خروجی متناظر این کلمه مقایسه شده

در Figure 38 کلمه C به عنوان ورودی داده شده و خروجی با بردار خروجی متناظر این کلمه مقایسه شده است. همانطور که دیده می‌شود، خروجی با خروجی متناظر ورودی مطابقت دارد.

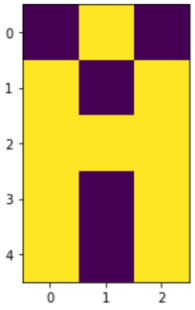


```
[ True  True  True]
True
```

Figure 38 کلمه **C** به عنوان ورودی داده شده و خروجی با بردار خروجی متناظر این کلمه مقایسه شده

در Figure 39 بردار $(-1,-1,-1)$ به عنوان ورودی داده شده و خروجی با حرف A مقایسه شده است. همانطور که دیده می‌شود، خروجی با A مطابقت دارد.

```
y:  [-1 -1 -1]
```

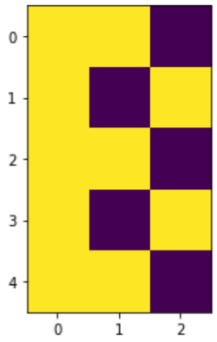


```
pred == ins[ 0 ] :  True
```

Figure 39 بردار $(-1,-1,-1)$ به عنوان ورودی داده شده و خروجی با حرف **A** مقایسه شده

در Figure 40 بردار $(-1,-1,1)$ به عنوان ورودی داده شده و خروجی با حرف B مقایسه شده است. همانطور که دیده می‌شود، خروجی با B مطابقت دارد.

```
y:  [-1 -1  1]
```

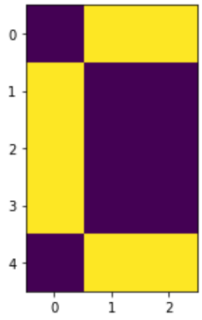



```
pred == ins[ 1 ] :  True
```

Figure 40 بردار $(-1,-1,1)$ به عنوان ورودی داده شده و خروجی با حرف **B** مقایسه شده

در Figure 41 بردار $(-1,1,-1)$ به عنوان ورودی داده شده و خروجی با حرف C مقایسه شده است. همانطور که دیده می‌شود، خروجی با C مطابقت دارد.

```
y:  [-1  1 -1]
```




```
pred == ins[ 2 ] :  True
```

Figure 41 بردار $(-1,1,-1)$ به عنوان ورودی داده شده و خروجی با حرف **C** مقایسه شده

همان طور که در بالا دیدیم، در تمام حالات تداعی شبکه برای هر حرف ورودی درست است. همچنین دیدیم برای هر حالت بردار خروجی، حرف تداعی شده با حرف متناظر آن بردار خروجی مطابقت دارد و شبکه در تمام حالات توانسته تداعی درستی داشته باشد. در این شبکه برای بازیابی زوج ورودی و خروجی از زوج ورودی و خروجی دارای نویز به صورت Figure 42 عمل می‌کنیم. برای این کار ابتدا y را در ترنسپوز ماتریس وزن‌ها ضرب می‌کنیم و از تابع فعالساز عبور می‌دهیم تا تداعی از ورودی یا xl بدست آید. بعد از بدست آمدن این تداعی، آن را با حروف اصلی مقایسه می‌کنیم. اگر با یکی از این حروف برابر بود، الگوریتم

به آن حرف و بردار خروجی متناظر آن همگرا شده است. چون با ضرب xl تداعی شده که یک حرف اصلی است در ماتریس وزن و عبور دادن از تابع فعالساز، ما به بردار خروجی متناظر حرف اصلی می‌رسیم. اگر xl حرف اصلی نبود، آن را در ماتریس وزن ضرب می‌کنیم و از تابع فعالساز عبور می‌دهیم تا yl یا تداعی بردار خروجی بدست آید. باز تداعی را با بردارهای اصلی مقایسه می‌کنیم. اگر یکی از بردارهای اصلی باشد ما به آن بردار و حرف متناظر آن همگرا شده ایم. اگر این طور نبود، ما یک مرحله از بازیابی را انجام داده ایم. اگر xl و yl با x و y ورودی یکی باشد، ما به xl و yl همگرا شده ایم. چون در مرحله بعد از xl و yl بدست آمده به جای x و y استفاده می‌کنیم و این باعث می‌شود که با انجام مرحله بعد دو باره به زوج تکراری تداعی ورودی و خروجی برسیم.

```python
def restore(x, y, w, ins, outs):
  xl = x.copy()
  yl = y.copy()
  yp = y.copy()
  xp = x.copy()

  while True:
    # print(xl, yl)
    temp = np.matmul(yl.flatten().reshape(1, -1), w.T)
    for i, p in enumerate(temp[0]):
      xl[0][i] = act(p, xl[0][i])

    res = find(xl, ins)
    if res >= 0:
      return xl, outs[res]

    temp = np.matmul(xl.flatten().reshape(1, -1), w)
    for i, p in enumerate(temp[0]):
      yl[0][i] = act(p, yl[0][i])

    res = find(yl, outs)
    if res >= 0:
      return ins[res], yl

    if np.all(yl.flatten().reshape(1, -1) == yp.flatten().reshape(1, -1)) or np.all(xl.flatten().reshape(1, -1) == xp
      return xl, yl

    yp = yl.copy()
    xp = xl.copy()
```

Figure 42 بازیابی زوج ورودی و خروجی از زوج ورودی و خروجی دارای نویز

اگر تداعی‌های ورودی و خروجی بعد از یک مرحله با ورودی و خروجی مرحله یکی نبود، با انجام مرحله بعد ادامه می‌دهیم. آنقدر مرحله تداعی انجام می‌دهیم تا یکی از شرط های همگرایی اتفاق افتد. در اینجا زوج x و y اولیه ورودی‌های شبکه هستند که توضیح آن در Figure 43 آمده است.

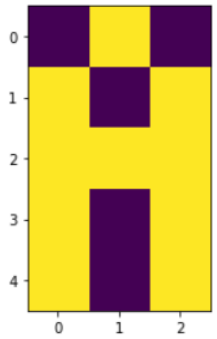
1. This Network is a recurrent form of Hetro-Associative Network
2. At First, two disturbed and deformed pattern of s(p) and t(p) are assigned to “x” and “y” as two entrances of the network.

Figure 43 زوج **x** و **y** اولیه ورودی‌های شبکه هستند

در اینجا این روش بازیابی اطلاعات بر اساس ماتریس وزن‌ها را روی حروف و بردارهای متناظر واقعی آنها اعمال می‌کنیم. در Figure 44 تداعی حرف و بردار خروجی برای A و بردار خروجی متناظر آن آمده است. همانطور که دیده می‌شود، تداعی با ورودی‌ها مطابقت دارد.

```
for i in range(len(ins)):
  y = outs[i].flatten().reshape(1, -1)
  x = ins[i].flatten().reshape(1, -1)
  px, py = restore(x, y, w, ins, outs)
  print('input : ', i)
  plt.imshow(px.reshape(5,3))
  plt.show()
  print(py)
```

```
input :  0
```

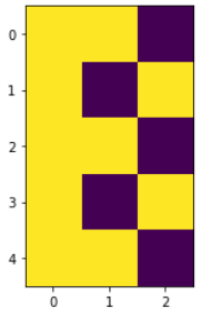



```
[-1 -1 -1]
```

Figure 44 تداعی حرف و بردار خروجی برای **A** و بردار خروجی متناظر آن

در Figure 45 تداعی حرف و بردار خروجی برای B و بردار خروجی متناظر آن آمده است. همانطور که دیده می‌شود، تداعی با ورودی‌ها مطابقت دارد.

```
input :  1
```




```
[-1 -1  1]
```

Figure 45 تداعی حرف و بردار خروجی برای **B** و بردار خروجی متناظر آن

در Figure 46 تداعی حرف و بردار خروجی برای C و بردار خروجی متناظر آن آمده است. همانطور که دیده می‌شود، تداعی با ورودی‌ها مطابقت دارد.

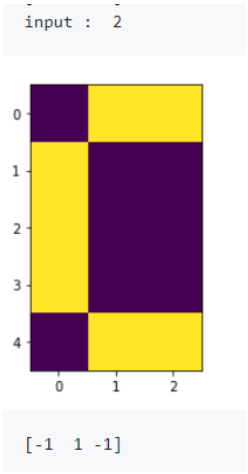


Figure 46 تداعی حرف و بردار خروجی برای **C** و بردار خروجی متناظر آن

همانطور که دیده شد، در تمام موارد تداعی شبکه با ماتریس وزن و این روش بازیابی اطلاعات برای حروف اصلی و بردار خروجی متناظر آنها، کاملا درست بود. این یعنی شبکه توانایی دارد با این روش اطلاعات را از ورودی به خروجی و بالعکس بازیابی کند.

ج

برای بدست آوردن درصد خروجی درست شبکه برای یک حرف و برای یک درصد نویز، به آن حرف و بردار خروجی متناظر آن مقدار نویز مشخص شده را به صورت راندم مانند Figure 7 اضافه می‌کنیم. سپس حرف و بردار خروجی نویز دار را برای بازیابی به شبکه می‌دهیم. روش محاسبه درصد موفقیت هر حالت نویز برای هر حرف در Figure 47 آمده است.

```
accs = []
for i in range(len(ins)):
  acc = 0
  for j in range(100):
    y = add_noise(outs[i].flatten().reshape(1, -1), 0.4)
    x = add_noise(ins[i].flatten().reshape(1, -1), 0.4)
    px, py =restore(x, y, w, ins, outs)
    acc += int(find(py, outs) >= 0)
  accs.append(acc/100)
```

Figure 47 روش محاسبه درصد موفقیت هر حالت نویز برای هر حرف

با توجه به Figure 47 مقدار درصد موفقیت برای حروف برای نویز ۴۰ درصد در Figure 48 آمده است. مقدار درصد موفقیت برای حروف برای نویز ۱۰ درصد در Figure 49 آمده است.

```
[0.54, 0.49, 0.54]
```

Figure 48 درصد موفقیت برای حروف برای نویز ۴۰ درصد

```
[0.68, 0.68, 0.66]
```

Figure 49 درصد موفقیت برای حروف برای نویز ۱۰ درصد

همانطور که در Figure 48 و Figure 49 دیده می‌شود، با افزایش نویز درصد موفقیت کاهش پیدا کرده است. درصد موفقیت A و B و C به ترتیب از چپ به راست در عکس‌ها آمده است. دلیل آن این است که با افزایش نویز تعداد بیشتری از درایه‌های حرف و بردار متناظر اشتباه است و به همین دلیل اطلاعات کمتری از حرف اصلی و بردار خروجی متناظر آن در دست داریم. این اطلاعات کمتر بازیابی حرف درست را سخت تر می‌کند و اگر نویز خیلی زیاد باشد، هیچ اطلاعی نداریم و درصد موفقیت بازیابی خیلی پایینی خواهیم داشت.

د

بله. این بردار را به عنوان y و بردار کاملا صفر را به عنوان x می‌دهیم. خروجی شبکه A است و توانسته برای این بردار خدجی تداعی داشته باشد. خروجی تداعی برای این بردار خروجی در Figure 50 آمده است.

```
px, py = restore(np.zeros((15)).flatten().reshape(1, -1), np.array([0,-1,-1]).flatten().reshape(1, -1), w, ins, outs)
```

```
print('pred y: ', py)
plt.imshow(px.reshape(5,3))
plt.title('pred x')
plt.show()
```

```
pred y:  [-1 -1 -1]
```

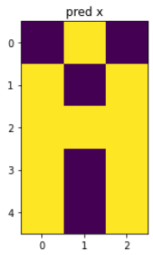


Figure 50 خروجی تداعی برای این بردار خروجی

دو درایه سمت راست بردار 1- است و در تمام حروف ورودی اصلی فقط بردار متناظر A اینگونه است. پس حرف متناظر بردار باید A باشد که شبکه به درستی تشخیص داده است.

۵

در اینجا مانند قسمت الف از تمام ۸ حرف برای ایجاد ماتریس وزن‌ها به روش هب مانند Figure 1 می‌پردازیم. ماتریس وزن‌ها برای ۸ حرف در Figure 51 آمده است.

```
[[ 2. -2.  6.]
 [-2. -2. -2.]
 [ 6.  2. -2.]
 [ 0.  0.  0.]
 [ 0.  0.  0.]
 [-4.  0.  4.]
 [ 0.  0.  0.]
 [ 2. -6.  2.]
 [ 0.  4.  0.]
 [ 0.  0.  0.]
 [ 0.  0.  0.]
 [-2.  2.  2.]
 [ 0. -4.  4.]
 [-2.  2. -2.]
 [ 2.  2. -6.]]
```

Figure 51 ماتریس وزن‌ها برای ۸ حرف

در Figure 51 می‌بینیم که تعداد قابل توجهی صفر در ماتریس وزن‌ها در این حالت وجود دارد. برای مثال ردیف ۴ و ۵ این ماتریس وزن‌ها صفر است. این یعنی دو خانه اول از سمت چپ دومین ردیف حرف ورودی در تعیین بردار ۳ تایی خروجی هیچ تاثیری ندارد. در ستون آخر می‌بینیم که خانه اول و آخر ردیف سوم حرف ورودی در تعیین درایه سوم بردار خروجی هیچ تاثیری ندارد. این موضوع در دو حرف A و B که در درایه سوم بردار خروجی فقط فرق دارند می‌تواند تاثیر گذار باشد. چون این دو حرف در خانه اول و آخر ردیف سوم تفاوت دارند. در Figure 52 این دو حرف A و B کنار هم آمده است.

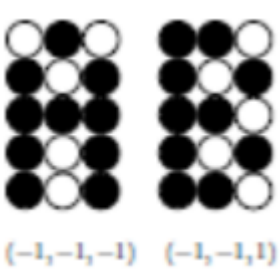
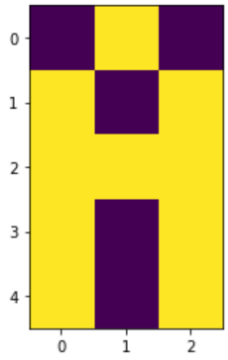


Figure 52 دو حرف **A** و **B** کنار هم

این موضوع می‌تواند باعث شود که شبکه با این ماتریس وزن‌ها نتواند بین این دو حرف تمایز خوبی برقرار کند.

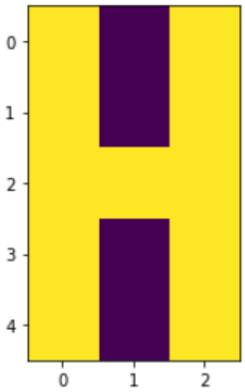
و

خیر. نمی‌تواند ۸ حرف را هم زمان ذخیره کرد. دلیل آن این است که شبکه حداکثر به اندازه مینیمم بین اندازه بردار ورودی یعنی ۱۵ و اندازه بردار ورودی خروجی یعنی ۳ می‌تواند پترن ذخیره کند. یعنی حداکثر می‌تواند ۳ پترن را ذخیره کند. برای بررسی توان ماتریس وزن شبکه برای ذخیره، تمام حروف را در ماتریس وزن ضرب می‌کنیم و خروجی را با بردار خروجی متناظر حرف مقایسه می‌کنیم. در Figure 53 و Figure 54 دو نمونه از ناتوانی تداعی بردار خروجی بر اساس حرف ورودی آمده است.




```
[ True  True False]
False
```

Figure 53 ناتوانی در تداعی بردار خروجی درست با ورودی حرف **A** بدون نویز

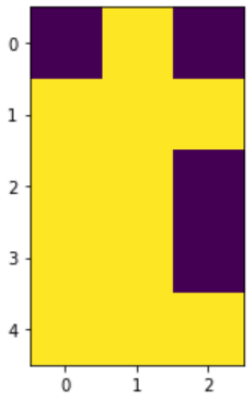



```
[ True False  True]
False
```

Figure 54 ناتوانی در تداعی بردار خروجی درست با ورودی حرف **H** بدون نویز

همانطور که دیده شد، برای تمام حروف شبکه با ماتریس وزن نمی‌تواند تداعی درستی داشته باشد. این یعنی نتوانسته‌ایم تمام حروف را ذخیره کنیم. همچنین توانایی شبکه را برای تداعی حرف بر اساس بردار خروجی را بررسی می‌کنیم. در Figure 55 و Figure 56 دو نمونه از تداعی حرف آمده است که کاملا اشتباه است.

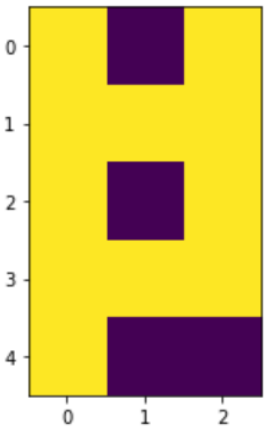
```
y:  [-1 -1 -1]
```




```
pred == ins[ 0 ] :  False
```

Figure 55 تداعی حرف بر اساس بردار خروجی متناطر حرف **A**

```
y:  [1 1 1]
```




```
pred == ins[ 7 ] :  False
```

Figure 56 تداعی حرف بر اساس بردار خروجی متناطر حرف **H**

برای تمام بردارهای خروجی حرف تداعی شده توسط ماتریس وزن شبکه اشتباه است. این موضوع نشان می‌دهد که نتوانسته ایم تمام حروف را هم زمان ذخیره کنیم.

ز

در Figure 57 فاصله همینگ دو به دوی حروف آمده است. حرفی که بیشترین فاصله همینگ از سایرین را داشته باشد، با احتمال زیاد با موفقیت ذخیره می‌شود. هر سطر ماتریس Figure 57 بردار فاصله آن حرف از سایرین است. هر حرف با اندازه بردار فاصله بیشینه می‌تواند به خوبی ذخیره شود.

	A	B	C	D	E	F	G	H
A	0	4	7	4	6	6	5	3
B	4	0	7	2	4	4	7	5
C	7	7	0	7	3	5	2	8
D	4	2	7	0	6	6	5	5
E	6	4	3	6	0	2	5	5
F	6	4	5	6	2	0	7	5
G	5	7	2	5	5	7	0	6
H	3	5	8	5	5	5	6	0

Figure 57 فاصله همینگ دو به دوی حروف

در Figure 58 توان دو اندازه بردار هر سطر در Figure 57 آمده است. حرف C مقدار بیشینه فاصله از سایرین را دارد و می‌تواند به خوبی ذخیره شود.

	norm^2
A	187
B	175
C	249
D	191
E	151
F	191
G	213
H	209

Figure 58 توان دو اندازه بردار هر سطر در ماتریس فاصله دو به دو