هوم ورک پنج‌ام شبکه عصبی .

مهدی فقهی
401722136

سوال اول :

الف) ایده کلی BAM را مختصرا شرح دهید.

یکی از مدل‌های بازگشتی مدل BAM است که یک مدل Biddirectional associative memory هست که دو تا الگو را به آن یاد می‌دهیم که وقتی یکی از الگوها را به آن بدهیم دیگری را فراخوانی می‌کند که دوتا توزیع متناظر را بهم ربط می‌دهد

ب) مدل BSB و Hopfield را با یکدیگر مقایسه کنید و در این مقایسه به بیان شباهت و تفاوت آن دو بپردازید.

شباهت‌ها :

هر دو این مدل از associative memories مدل استفاده می‌کنند .

هر دو از شیوه سعی می‌کنند الگو ورودی به به نزدیک‌ترین حالتی که تابع انرژی را منیمایز میکنند حرکت بدهند و الگو یادگیری مشابه را بیرون دهند .

شیوه یادگیری هر دو مدل یکسان هست .

تفاوت‌ها :

اما در BSB ما یک مدل fully connected داریم سلف کانکشن داریم و وزن‌ها متقارن نیست.

دیگر در BSB مدل نمی‌توانند در وضعیت میانی قرار بگیرد وحالت پایدار شبکه به شدت میرود در حالت‌های گوشه قرار می‌گیرد ولی هاپفیلد می‌تواند در حالت‌های میانی قرار بگیرد که این عمر باعث می‌شود که پایداری در BSB خیلی بهتر باشد .

در نتیجه می‌توان گفت BSB نسبت به noise حتی از hopfield هم بهتر عمل می‌کند و مدل درست را بهتر بازیابی کند .

ج) ضعف شبکه Hopfield در مقابل Boltzmann چی ست؟ و Boltzmann چگونه بر این مشکل غلبه میکند؟

مشکل مدل Hopfield در حافظه‌های ترکیب خطی یا splin galss می‌افتاد که به آن local minima می گفتیم و ما می‌خواهیم از این نقاط بیرون بپریم . در مدل Boltzman به کمک متد simulate annealing استفاده می کنیم اول ابتدا از یک دما بالا استفاده می کنیم کل حالت های ممکن را بهش امکان می‌دهیم که ببیند و بعد یکم یکم سرد میکنم تا در این نقطه بهترین نقطه‌ای که می‌تونست قرار بگیره بهترین جاش قرار بگیرد یعنی به کمک این روش از local minimam ها فرار می کنیم .

سوال دوم :

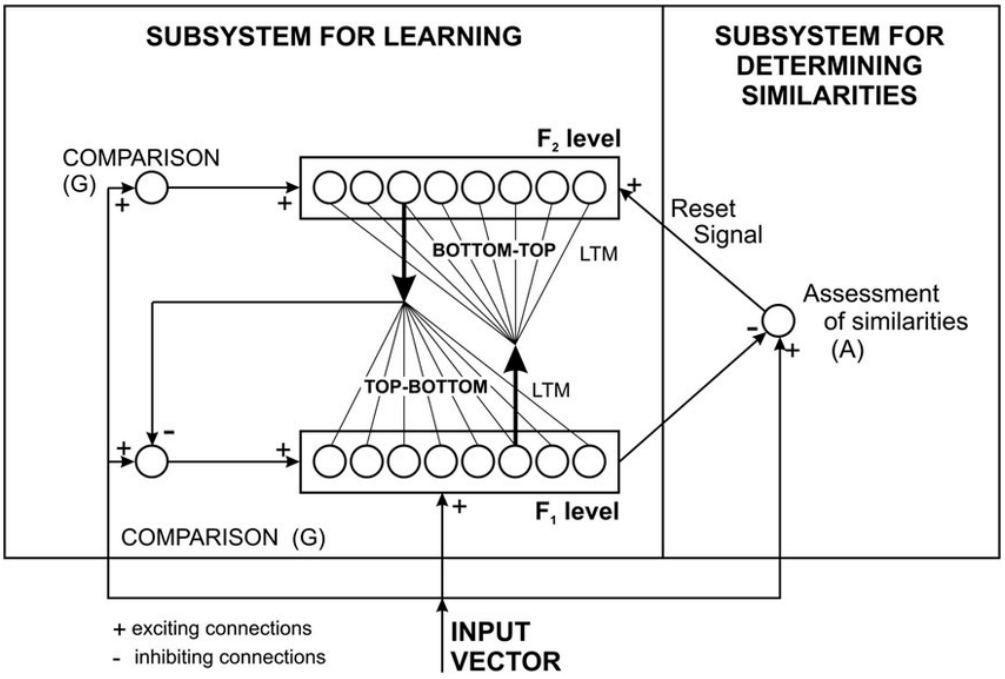
توضیح سوال دوم :

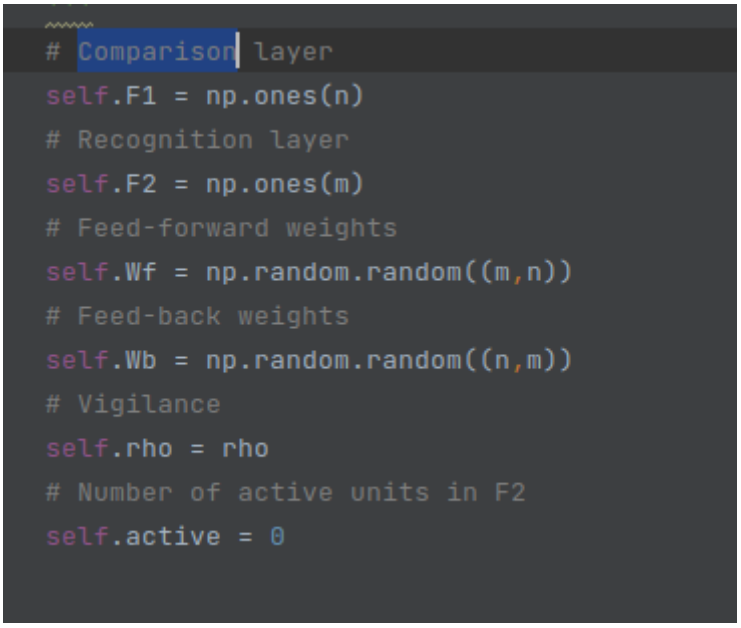
برای learn مدل از این کد آماده در github استفاده کردم که در پایین روند کاری این کد را توضیح می‌دهم .

https://github.com/rougier/neural-networks/blob/master/art1.py

یک کلاس به نام کلاس Art درست می کنیم با پارامتر ورودی n میزان ورودی مسئله ما و m ما کزیمم تعداد کلاسی که می‌توانیم به کمک مدل خودمون پیدا کنیم و rho که همان میزان شباهت را مشخص می کند .

سپس به تعریف لایه‌های موجود در معماری مدلمان پرداخته‌ایم .





```
# Comparison layer
self.F1 = np.ones(n)
# Recognition layer
self.F2 = np.ones(m)
# Feed-forward weights
self.Wf = np.random.random((m,n))
# Feed-back weights
self.Wb = np.random.random((n,m))
# Vigilance
self.rho = rho
# Number of active units in F2
self.active = 0
```

میدانیم لایه F1 به تعداد ورودی ما نرون و لایه F2 به تعداد حداکثر میزانی که خروجی کلاس می‌توانیم داشته باشیم خروجی دارد .

کانکشن بین لایه ورودی F1 و F2 به صورت fully connected و دو طرفه هست از سمت لایه F1 به F2 یک ماتریس وزن Wf تعریف کردیم و از سمت سمت F2 به F1 هم یک ماتریس وزن Wb که هر دو را به صورت رندوم بین صفر تا یک initial کرده ایم .

و تعداد خروجی‌های فعال کلاس را با active نشان می‌دهیم .

```
def learn(self, X):
    ''' Learn X '''

    # Compute F2 output and sort them (I)
    self.F2[...] = np.dot(self.Wf, X)
    I = np.argsort(self.F2[:self.active].ravel())[::-1]
    for i in I:
        # Check if nearest memory is above the vigilance level
        d = (self.Wb[:,i]*X).sum()/X.sum()
        if d >= self.rho:
            # Learn data
            self.Wb[:,i] *= X
            self.Wf[i,:] = self.Wb[:,i]/(0.5+self.Wb[:,i].sum())
            return self.Wb[:,i], i

    # No match found, increase the number of active units
    # and make the newly active unit to learn data
    if self.active < self.F2.size:
        i = self.active
        self.Wb[:,i] *= X
        self.Wf[i,:] = self.Wb[:,i]/(0.5+self.Wb[:,i].sum())
        self.active += 1
        return self.Wb[:,i], i
```

در قسمت learn همانطور که مشاهد می کنید برای learn کردن یک ورودی می گیریم که این ورودی وارد سیستم ما می‌شود سپس به کمک آیتم ماتریس وزن forward weights یک ضربه نقطه در آن می کنیم و به تعداد نرون‌های فعال یک لیست به دست می‌آوریم که نشان می‌دهد به ازای هر نرون F2 به چه میزان بزرگی رسیده‌ایم البته کلاس‌هایی که هنوز فعال نشده اند را از دایره این لیست کنار می گذاریم و براساس بزرگی آن‌ها اندیس‌های آن‌ها را در لیست I خواهیم داشت یعنی اگر بعد از ضرب لیست [2, 3, 10 , 4] را داشته باشیم بعداز argsort [3,2,0,1] را خواهیم داشت که با عمل [-1::] به شکل [1,0,2,3] خواهد شد و index آیتم با بزرگترین میزان که در اینجا ۱۰ است در ابتدا خواهد گرفت .

سپس براساس این index ها که براساس بزرگی ضرب داخلی بالا بدست آمده بود میزان شباهت بین ماتریس وزن backward را با ماتریس ورودی حساب می کنیم و در d قرار می‌دهیم و اگر این میزان بیشتر از rho ما باشد ماتریس backward وforward در ردیف مربوط به کلاس خروجی برنده آپدیت می کنیم . براساس فرمول‌های بالا. در قسمت backword ، لایه مذکور کلاس را با ورودی and و آپدیت می کنیم .

حال اما اگر هیچ یک از کلاس‌های فعال ما شبیه به ورودی نباشد باید یک کلاس جدید بسازیم و به تعداد کلاس‌های فعال ما یکی اضافه می‌شود .

در قسمت بعد به کلاس مربوط به یک فانکشن به نام prediction اضافه کردم که براساس ورودی ما نزدیکترین خروجی را براساس کلاس‌هایی که داریم خروجی می‌دهد .

```
def predict(self, X):
    ''' Test X '''
    # Compute F2 output and sort them (I)
    self.F2[...] = np.dot(self.Wf, X)
    I = np.argsort(self.F2[:self.active].ravel())[::-1]

    bigest = -1
    bigest_index = 0
    for i in I:
        # Check if nearest memory is above the vigilance level
        d = (self.Wb[:,i]*X).sum()/X.sum()
        if bigest > d :
            bigest_index= i
            bigest = d


    return self.Wb[:,bigest_index], bigest_index
```

در این قسمت همانند بالا عمل می کنیم با این تفاوت که دیگر از میزان شباهت استفاده نمی کنیم و فقد نگاه می کنیم کدام کلاس بیشترین شباهت را به مدل موردنظر دارد و آن را برمیگردانیم .

خوب از آنجا که شماره دانشجوی من رقم یکانش ۶ بود و بر ۳ بخش پذیر بود پس باید مدل اول را یاد میگرفتم پس ورودی مدل اول را ساختم .

```
set_one_train = {"A" : [[0,1,1,0],
                        [1,0,0,1],
                        [1,1,1,1],
                        [1,0,0,1]],

                 "C":[[0,1,1,1],
                      [1,0,0,0],
                      [1,0,0,0],
                      [0,1,1,1]],

                 "J":[[0,1,1,1],
                      [0,0,1,0],
                      [1,0,1,0],
                      [1,1,1,0]],

                 "L":[[1,0,0,0],
                      [1,0,0,0],
                      [1,0,0,0],
                      [1,1,1,1]]}
set_one_test = {"O":[[0,1,1,0],
                     [1,0,0,1],
                     [1,0,0,1],
                     [0,1,1,0]]}
```

سپس یک مدل با ۱۶ ورودی و با حداکثر ۱۰ کلاس در نظر گرفته ام با میزان شباهت ۲۵ درصد .

```
network = ART( 4*4, 10, rho=0.25)
dic_main = {}
```

بعد به گونه زیر عمل کردم تا وقتی که احساس کنم مدل من به یک حالت stable براساس داده ورودی رسیده است .

```
# we use learning until all our weight not change.
while True:
    dic_helper = {}
    for i in set_one_train:
            # Z is matrix weight of class k
            Z, k = network.learn(np.array(set_one_train[i]).ravel())
            dic_helper[k] = Z

    if not two_dic_is_same(dic_main,dic_helper):
        break
    else:
        dic_main = dic_helper
```

همه ورودی ها را به مدل می‌دهم و سپس یک k که شماره کلاس و z که ماتریس وزن backward ماست که میزان شباهت را براساس آن بدست می‌آوریم برای کلاس k را برمیگردانیم و یک دیکشنری از کلاس و ماتریس و وزن آن می‌سازم و این کار تا زمانی که دیکشنری کلاس و وزن تغییر نکند ادامه می‌دهم .

سپس در پایین آمده آمده ام به ازای هر ورودی ماتریس وزن و کلاس آن را نشان داده ام .

```
A-> class 0
[[0.         0.94833274 0.4942051  0.        ]
 [0.28470968 0.         0.         0.        ]
 [0.56867322 0.         0.         0.        ]
 [0.         0.         0.         0.28458536]]
C-> class 0
[[0.         0.94833274 0.4942051  0.        ]
 [0.28470968 0.         0.         0.        ]
 [0.56867322 0.         0.         0.        ]
 [0.         0.         0.         0.28458536]]
J-> class 1
[[0.         0.         0.         0.        ]
 [0.         0.         0.         0.        ]
 [0.77157846 0.         0.         0.        ]
 [0.21880439 0.97117124 0.89173377 0.        ]]
L-> class 1
[[0.         0.         0.         0.        ]
 [0.         0.         0.         0.        ]
 [0.77157846 0.         0.         0.        ]
 [0.21880439 0.97117124 0.89173377 0.        ]]
```

همانطور که مشاهده میکنید ورودی A , C به کلاس صفر و ورودی J , L به کلاس 1 مربوط شده اند .
و در ادامه به کمک function prediction کلاس O به کلاس صفر انتصاب داده شد .

```
# see class of input test
for i in set_one_test:
        Z, k = network.predict(np.array(set_one_test[i]).ravel())
        print(f"{i}-> class {k}")
        print(Z.reshape(4,4))
```

```
O-> class 0
[[0.         0.94833274 0.4942051  0.        ]
 [0.28470968 0.         0.         0.        ]
 [0.56867322 0.         0.         0.        ]
 [0.         0.         0.         0.28458536]]
```

دهید.در ادامه نمونه دادگان تست را به شبکه دهید تا عمل طبقه بندیانجام شود. آیا مدل شما قادر است درالگوهای پیوسته (Patternscontinuous)به خوبی کار کند؟ علت را مختصرا شرح دهید.

خیر . زیرا وقتی نمونه‌ها یادگیری به صورت پشت سر هم تنها در یک قسمت کوچیک هر کدام نسبت به دیگری تفاوت داشته باشند با در نظر گرفتن میزان شباهت زیاد ممکن است دوچار مشکل گره مادر بزرگ شویم و با در نظر گرفتن میزان شباهت کم یک کلاس تا میزان کاهش شباهت کوچیک می‌شود و داده‌ها را درون خود جای می‌دهد ولی از یکجا به بعد یک کلاس دیگر بوجود می‌آید اما آخرین داده‌ای که درون کلاس موردنظر قرار گرفته است با داده بعد که یک کلاس جدید می‌سازد شباهت بیشتری دارد تا نمونه های اولیه که در کلاس آنان قرار گرفته . سعی ما این است که با rho مناسب این مشکل را حل کنیم ولی خوب در نهایت برای هر rho این اشکال وجود دارد تنها راه مقابله با این موضوع این است که داده ها به صورت مناسب shuffle شوند تا داده‌ها نزدیک هم تا حد خوبی از هم دور شوند و پشت سر هم نباشند اینگونه احتمال اینکه داده‌های که بیشتر شبیه به هم هستن در یک کلاس قرار بگیرند بیشتر است .

3 -بعد از آموزش شبکه RCE کدام یک از موقعیتهای زیر ممکن است که رخ دهد؟ جواب بله یا خیر خود را مختصرا شرح دهید.

الف) داشتن دایره های هم مرکز، مربوط به یک کلاس یکسان.

خیر امکان ندارد زیرا مرکز هر دایره روی مختصات نقطه ورودی قرار می‌گیرد و اگر دو نقطه یکسان مربوط به یک کلاس روی هم قرار بگیرند به این معنا هست که یکی زودتر آمده است و شعاعی همسایگی خود را مشخص کرده است و دومی با قرار گرفتن روی آن نقطه عملا در فضای همان دایره است هست و نیاز به کشیدن دوباره خط شعاع ندارد .

ب) داشتن دایره های هم مرکز، مربوط به کلاس‌های متفاوت.

اگر وجود خطا طبیعی در دادگان آموزشی را پذیرفته باشیم امکان پذیر است در غیر این صورت خیر امکان پذیر نیست بسیار به رویکرد ما در حل بستگی دارد و به معنای وجود خطا در دادگان آموزشی است که جز فرض ما نیست .

خطای Byse مسئله ما نیز از وجود همچین اتفاقاتی سرچشمه می‌گیرد و ما به اندازه فضایی همپوشانی مکان‌هایی در فضا داریم که به طور صد در صد نمی‌توانیم به یکی از کلاس‌ها انتصاب بدیم .

البته مسئله‌ای هم وجود دارد اگر داده‌های ورودی ما دارای خطای ذاتی باشد و اگر ما کاهش شعاع همسایگی را تاجایی ادامه بدهیم که نقطه با کلاس متفاوت در کلاس ما نباشد این کار باعث می‌شود در یک لوپ بینهایت بی‌افتیم به دلیل کاهش شعاع همسایگی یا به شعاع همسایگی صفر که باید دید در پیاده سازی ما چگونه خطای ذاتی مسئله یا byes را هندل می‌کنیم و اگر ما فقط پذیرفته باشیم داده‌هایی که خطا ندارند را train کنیم توانایی کار روی همچین مسئله‌نداریم ولی اگر هنگام برخورد با همچین رویکردی ناحیه اطراف این دو نقطه را به اندازه یک مقدار کم جز نواحی که قابلیت تعمیم نداریم شناسای کنیم امکان پذیر است .

ج) داشتن دایرههای مماس، مربوط به یک کلاس یکسان.

بله این اتفاق هم امکان پذیر است شعاعی که برای نقطه اول در نظر گرفته‌ایم به اندازه کافی بزرگ نیست و با ورود نقطه بعدی در محیطی قرار میگیرد که قبلا تعریف نشده است در نتیجه یک دایره با شعاع بوجود میارد که آن هم به اندازه کافی بزرگ نیست که دایره دیگر را در خود در برگیرد و به صورت شانسی مماس می‌شوند .

د) داشتن دایرههای مماس، مربوط به کلاس‌های متفاوت.

بله این اتفاق کاملا طبیعی امکان پذیر است . و از پدید امدن ناحیه‌ای که توانایی انتصاب نقطه تست در آن به هیچ کلاس دیگر نداریم را کاهش می‌دهد.

ه) داشتن دایرهای که توسط دایره دیگر محصور شده است

بله این اتفاق هم امکان پذیر است برای مثال دو دایره برای فضای مشکی داریم که در ابتدا شعاع اولی را کوچکتر گرفته ایم در ادامه با آمدن نقطه دوم و انتصاب شعاع بزرگتر به آن این امکان وجود دارد که دایره اول به طور کامل توسط دایره دوم پوشیده شود .

4 -به سواالت زیر پاسخ دهید.

الف) به نظر شما، چرا شبکه Art زمانی که نویز زیادی داشته باشیم، عملکرد ضعیفی را ارائه خواهد داد؟ (ارائه یک تحلیل منطقی (و شاید نادرست) کفایت میکند.)

چون هنگامی که نویز زیادی داشته باشیم کلاس‌هایی که باید در یک کلاس یکسان قرار بگیرند به دلیل افزایش noise میزان شباهتشان نسبت به همدیگر کاهش پیدا می کند در نتیجه در کلاس‌هایی جدا از هم کلاس بندی می‌شوند که این مطلوب ما نیست .

ب) مشکل " گره مادر بزرگ" (node-Grandmother)را در شبکه‌های Art شرح دهید.

اگر میزان p یا (رو) را بالا بگیریم و در میزان شباهت سختگیری به عمل بیاریم ممکن هست یک ورودی خودش به تنهایی تبدیل به یک کلاس شود که باعث می‌شود تعداد کلاس‌های ما زیاد شود و این مطلوب ما نیست که به ازای هر ورودی یک کلاس دسته بندی داشته باشیم .

این باعث میشه مدل ما شبیه Brain نباشه و اگر یکی از cellها خراب شود نسبت به آن robust نیست و کل کلاس از بین میره و آنها فقط می‌توانند تعداد محدودی از clusterها را در خود نگه دارند .

ج) میدانیم که در RCE ،در مجموعه دادگان آموزش، هیچ خطایی نداریم ولی امکان وجود خطا در تست وجود دارد. به نظر شما، علت عدم وجود خطا در هنگام آموزش و احتمال وجود خطا در تست چیست؟

در هنگام آموزش ما به ازای هر نمونه که وارد فضای ما می‌شود بررسی می کنیم که این نمونه در چه وضعیتی از صفحه ما قرار گرفته است . که سه حالت داریم : ۱ ـ یا در یک دایره با label کلاس خودش قرار گرفته است ۲ـ یا در کلاس مخالف قرار گرفته است یا ۳ـ در فضایی بیرون از این دو فضا قرار گرفته است .

برای حالت اول کار خاصی انجام نمی‌دهیم ولی برای حالت دوم و سوم یک شعاع به اندازی که داده با label مخالف را در دایره ما قرار ندهد انتخاب می کنیم . پس هیچ خطایی نمی‌توانند در هنگام آموزش رخ دهد .

و از آنجایی که ما وجود هرگونه خطا را در مجموعه دادگان آموزش را محال فرض کرده‌ایم یعنی حتی خطای byes را هم شاید وجودش را در نظر نگرفته‌ایم هیچ خطای ذاتی برای مسئله ما وجود ندارد در هنگام train نیز پس کلا بدون خطا هستیم.

اما بسیار پیش میاید که در هنگام تست یک نقطه در فضا و شعاع label ای قرار بگیرد که متعلق به آن نیست و اینگونه خطا رخ می‌دهد که ممکن است در اثر میزان کم داده برای train مدل باشد یا overfit شدن بیش از اندازه مسئله ما روی داده‌های محدود train و هر چقدر از قدرت generality مدل ما کاهش پیدا کند احتمال وجود خطا در مسئله test افزایش پیدا می‌کند .

پس باید سعی کنیم تا جایی که می‌شود تعداد دادگان با پراکندگی مناسب از فضای نمونه را در اختیار حالت train بگذاریم که در نتیجه آن خطای تست ما کاهش یابد .