به نام خدا

گزارش تمرین دوم مبانی هوش

غزل زمانی نژاد، 97522166

1. الف) در این سوال می خواهیم شبکه SOFM را پیاده سازی کنیم. این روش یادگیری، روشی Self organized است. در آن پس از اتمام یادگیری، وزن ها بر اساس مقادیر ورودی به گونه ای مقدار می گیرند که داده های شبیه تر به یکدیگر نزدیک تر باشند.

ابتدا کلاس Kohonen را پیاده سازی کرده و سپس از آن استفاده می کنیم. در constructor این کلاس تعداد دیتاهای ورودی، اندازه نقشه Kohonen، دیتای ورودی، نرخ یادگیری، شعاع همسایگی و تعداد epoch را به عنوان ورودی دریافت می کنیم. جهت training بهتر، دیتاها را نرمالایز می کنیم. (در اینجا بیشترین مقدار ورودی 255 است، پس همه ورودی ها را بر این مقدار تقسیم می کنیم.)
سپس با توجه به ابعاد شبکه Kohonen، تعدادی وزن تعریف کرده و آن ها را به صورت رندوم initialize می کنیم. چون برای مشخص کردن هر رنگ از 3 مقدار r، g و b استفاده می شود، برای هر نورون باید 3مقدار وزن مربوط به هر رنگ در نظر بگیریم.
مرحله بعد از تشکیل شبکه، competition است. یکی از ورودی ها را به صورت رندوم انتخاب کرده، و سپس میزان فاصله اقلیدسی آن را با همه وزن های موجود در شبکه محاسبه می کنیم. ایندکسی که وزن آن کمترین فاصله (یعنی بیشترین شباهت) را با داده موجود داشته باشد به عنوان winner برمی گردانیم.

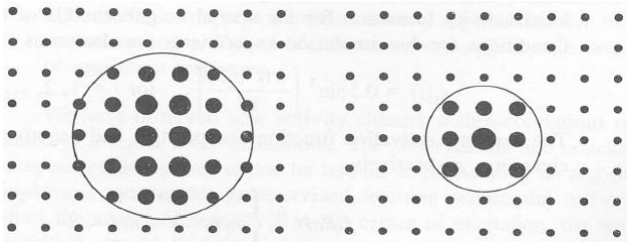
The Euclidian distance: $i(x)=\arg\min ||X-W_j||$

در مرحله سوم، تابع همسایگی تعریف می کنیم. در اینجا از تابع همسایگی گاوسی با فرمول زیر استفاده شده.

$$h_{j,i(x)} = e^{\frac{-d_{j,i}^2}{2\sigma^2}}$$

در این فرمول d نشان دهنده فاصله اقلیدسی نورون برنده تا سایر نورون هاست.

هم چنین sigma، نشان دهنده شعاع تابع همسایگی است. مطابق تصویر زیر، هرچه بیشتر باشد، نورون های بیشتری را تحت تاثیر خود قرار می دهد.

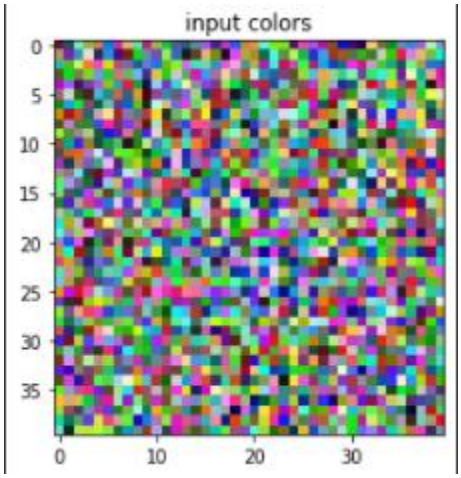


در پیاده سازی ابتدا برای سادگی کار تابعی تعریف شده که ایندکس هر نورون روی نقشه را برمی گرداند. (تابع indices_array) سپس در تابع distToWinner مقدار فاصله همه نورون ها تا نورون برنده طبق فرمول اقلیدسی محاسبه شده است.

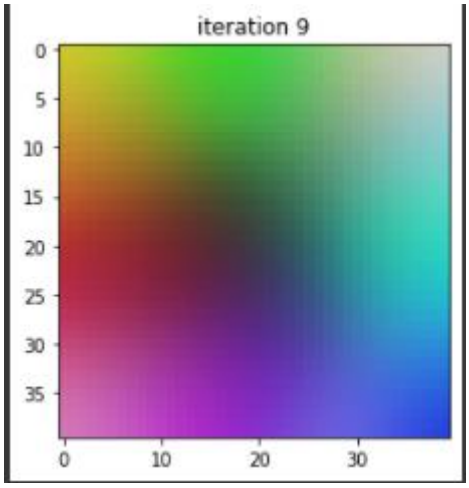
در مرحله بعد باید وزن ها را آپدیت کنیم. در متد changes، از تابع همسایگی که تعریف کردیم استفاده می کنیم و مقدار آن را در h ذخیره می کنیم. سپس مقدار وزن ها را از دیتای ورودی کم کرده، در learning_rate و h ضرب می کنیم. با اضافه کردن مقدار بدست آمده به وزن ها، آن ها را آپدیت می کنیم.

$$\Delta w_j = \eta h_{j,i(x)}(x - w_j)$$

اکنون می خواهیم شبکه را تست کنیم. ابتدا در ورودی 1600 رنگ به صورت وارد می کنیم. (مقدار هر یک از rgb باید بین 0 تا 255 باشد). این رنگ ها را در یک نقشه 40 * 40 با استفاده از متد imgshow (از کتابخانه pylab) که مربوط به کشیدن تصاویر است پلات می کنیم.



سپس یک instance از کلاس kohonen با 1600 ورودی، ابعاد 40 * 40، ورودی رنگ ها، نرخ یادگیری 01 و تعداد epoch 10 می سازیم. و بر روی آن تابع learn را صدا می زنیم. در این تابع 10 بار روی همه دیتاها iterate می کنیم. در هر یک از epoch ها روی تک تک داده ها، ابتدا تابع competition را صدا می زنیم تا نورون برنده را مشخص کنیم. سپس وزن ها را آپدیت می کنیم. پس از پایان هر epoch، وزن های آپدیت شده را در Kohonen map چاپ می کنیم. نتایج به صورت زیر است.

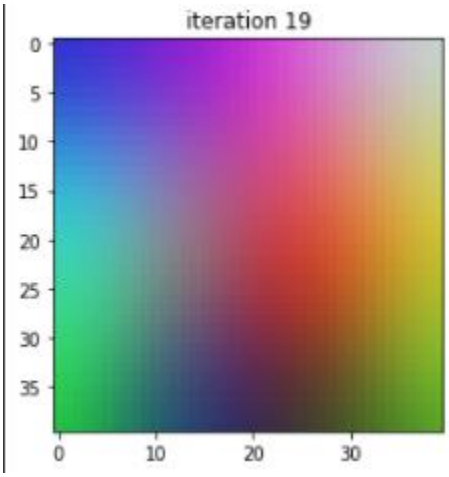


مطابق انتظار، وزن هایی از شبکه که شباهت رنگ بیشتری داشته اند در کنار هم قرار گرفته اند.

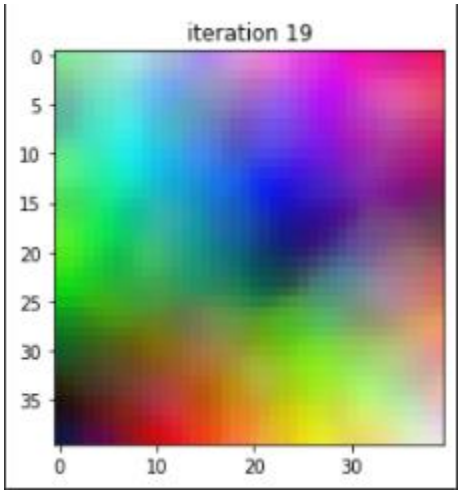
ب) این سوال مشابه سوال قبل پیاده سازی شده است. تنها تفاوت آن، استفاده از متد decrease_lr است. در این متد، نرخ یادگیری بعد از هر epoch به صورت نمایی تغییر می کند. و از learning rate جدید برای محاسبه میزان آپدیت وزن ها در هر epoch استفاده می شود.

هنگامی که از ضریب یادگیری ثابت استفاده می کنیم، ممکن است پروسه یادگیری هیچ وقت متوقف نشود و در هر iteration، رنگ ها جا به جا می شوند. در سوال قبل پس از هر iteration می توانیم جابه جایی رنگ ها را مشاهده کنیم. به این پدیده principal curves گفته می شود. principal curves منحنی های یک بعدی هستند که از میان داده ها را به صورت غیرخطی عبور می کنند.

اما پس از کاهش ضریب یادگیری در پایان هر epoch، تغییر کمتری در نقشه kohonen دیده می شود. همچنین می توانیم با کاهش این ضریب از convergence اطمینان حاصل کنیم.



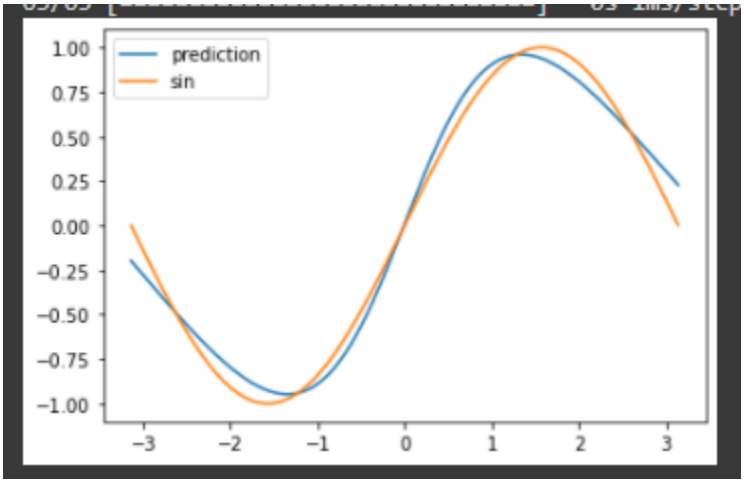
ج) در این قسمت، متد decrease_radius را به پیاده سازی قبل اضافه می کنیم. در این متد، شعاع تابع همسایگی بعد از هر epoch به صورت نمایی تغییر می کند. با کاهش شعاع، در iterationهای مختلف مشاهده می شود که تفاوت رنگ ها بیشتر قابل رویت است.



2. الف) در این سوال با استفاده از کتابخانه keras می خواهیم از یک MLP برای یادگیری تابع سینوس استفاده کنیم. ابتدا MLP را با استفاده از تابع sequential می سازیم. لایه های آن به صورت زیر است:
   - در لایه اول، 50 نورون وجود دارد و از تابع فعالسازی sigmoid استفاده شده است.
   - در لایه دوم که همان لایه hidden می باشد، 30 نورون وجود دارد و از تابع فعالسازی sigmoid استفاده شده است.
   - و در لایه آخر یک نورون برای یافتن مقدار خروجی وجود دارد.

   پرسپترون تشکیل شده را با استفاده از stochastic gradient descent و تابع ارور MSE کامپایل می کنیم. سپس به آن در 50 epoch و 10 bacth آموزش می دهیم. (epoch: تعداد iteration بر روی کل داده ها، batch: تعداد نمونه هایی که قبل از آپدیت روی آن ها یادگیری انجام می شود.)

   پس از اتمام یادگیری، با استفاده از تابع predict، پرسپترون را تست می کنیم. (در این سوال دیتای تست همان دیتای یادگیری است.) و در پایان تابع سینوس را در کنار مقادیری که پرسپترون پیش بینی کرده پلات می کنیم. و به آنها label می دهیم. نتیجه در زیر مشاهده می شود:



ب) در این قسمت به پیاده سازی RBF می پردازیم. برای این کار ابتدا یک کلاس RBF تعریف می کنیم. در constructor آن، مقدار کرنل (تعداد توابع radial)، ضریب یادگیری و تعداد epochها را به عنوان ورودی دریافت می کنیم. سپس وزن ها و biasها را به صورت رندوم initial می کنیم. و دو لیست خالی برای ذخیره سازی مرکز و شعاع توابع Radial تعریف می کنیم.

این الگوریتم، به MLP شباهت دارد. تفاوت آن با MLP در این است که برای هر یک از نورون های لایه میانی radial basis function تعریف می کنیم. برای این کار ابتدا باید مرکز و شعاع هریک از این توابع را تعیین کنیم. این کار را با استفاده از الگوریتم k-means clustering انجام می دهیم.

Algorithm 1 k-means algorithm

1: Specify the number k of clusters to assign.
2: Randomly initialize k centroids.
3: **repeat**
4: **expectation:** Assign each point to its closest centroid.
5: **maximization:** Compute the new centroid (mean) of each cluster.
6: **until** The centroid positions do not change.

در ابتدا به پیاده سازی تابع k_cluster می پردازیم. به تعداد کرنل از میان داده های ورودی به صورت تصادفی انتخاب می کنیم و آن ها را به مرکز clusterها assign می کنیم. اکنون در یک loop، مراکز clusterها را تغییر می دهیم. شرط خروج از لوپ، converge است. یعنی میزان تغییر مرکز هر cluster نسبت به iteration قبل بسیار ناچیز باشد. در یک while loop، میزان فاصله هر یک از دیتاهای ورودی را نسبت به تمامی مراکز می یابیم. هر داده به clusterای متعلق است که کمترین

میزان فاصله را با آن دارد. پس با استفاده از arg min، cluster که به آن تعلق دارد پیدا می کنیم. بعد از انجام این کار بر روی همه داده ها، مراکز clusterها را آپدیت می کنیم تا دیتاها دقیق تر در clusterها قرار گیرند. برای هر مرکز، میانگین نقاطی که به آن cluster تعلق دارند را می یابیم و به عنوان مقدار جدید مرکز قرار می دهیم. بعد از آپدیت تمامی مرکزها شرط convergence را چک میکنیم. با برقراری این شرط، لوپ تمام می شود.

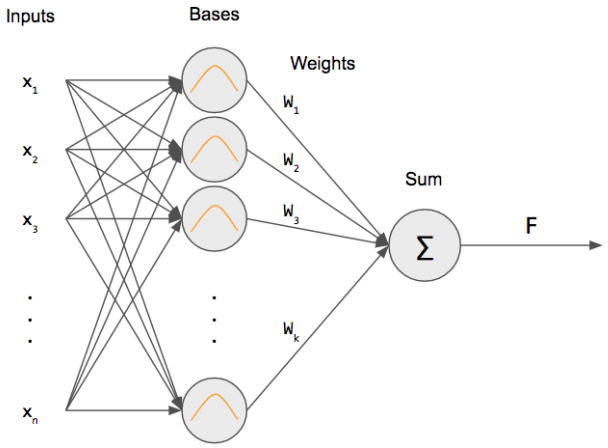
اکنون شعاع این clusterها را پیدا می کنیم. برای هریک از RBFها، شعاع را مقدار انحراف معیار نقاط متعلق به آن cluster درنظر می گیریم. اما برای clusterهایی که کمتر از دو نقطه به آن تعلق دارند، شعاع را مقدار میانگین شعاع clusterهای دیگر در نظر می گیریم. در خروجی، مراکز و شعاع clusterها بازمی گردانیم.

در مرحله بعد، توابع فعال سازی را guassian radial basis تعریف می کنیم:

$$\phi_i(\mathrm{x}) = \exp\left(-\frac{\|\mathrm{x}-t_i\|^2}{\sigma_i^{\,2}}\right)$$

در این تابع t مختصات مرکز و sigma شعاع cluster می باشد.

در مرحله بعد، همانند MLP به training می پردازیم. ابتدا مراکز clusterها را می یابیم. بعد به تعداد epoch روی تمامی داده ها iterate می کنیم. در هر iteration باید روی تمامی داده iterate کنیم.



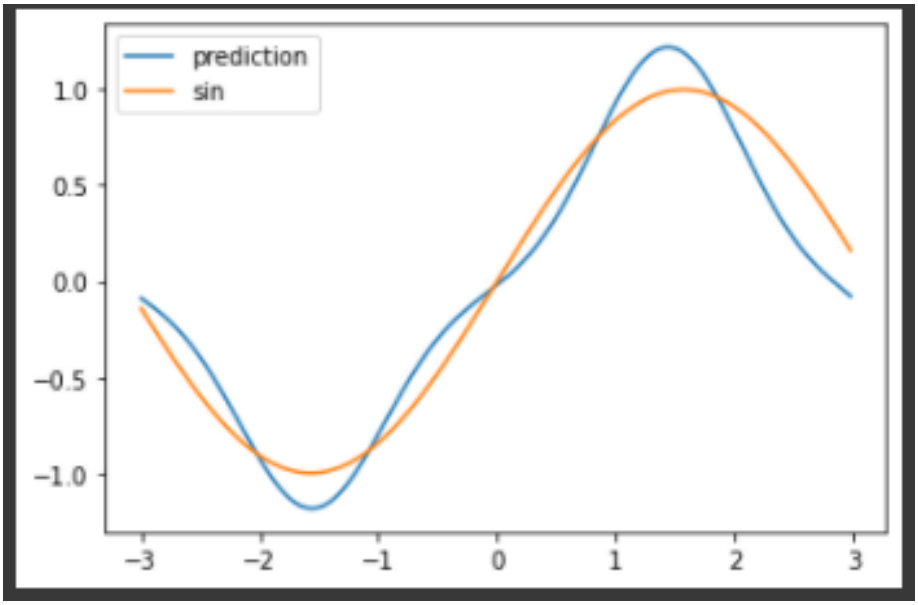
- در مرحله forward pass، داده را به تک تک توابع Radial می دهیم و مقدار آن را ذخیره می کنیم. مقادیر را در وزن ها ضرب کرده و با هم جمع می کنیم. مقادیر bias را نیز اضافه می کنیم.

$$\hat{F}(x) = \sum_{i=1}^{N} w_i . \exp\left(-\frac{\|x - t_i\|^2}{\sigma_i^2} \right) \cong F(x)$$

- در مرحله back propagate، ارور را با تابع mean squred error می یابیم. یعنی مقدار بدست آمده را از مقدار واقعی کم کرده و به توان 2 می رسانیم.
- در پایان مقدار وزن ها را آپدیت می کنیم.

در پایان نوبت به تست این شبکه عصبی می رسد. در تابع predict، بر روی تمامی دیتاهای تست iterate می کنیم. آن را به تمامی RBFها می دهیم و مقدار را ذخیره می کنیم. سپس در تمامی وزن ها ضرب می کنیم و با هم جمع می کنیم. این مقادیر را ذخیره کرد و به عنوان خروجی بازمی گردانیم.

پس از پیاده سازی کلاس، شبکه عصبی را تست می کنیم. آن را در بازه 6- تا 6 آموزش می دهیم. جهت robustness، مقداری را به عنوان noise به مقادیر سینوس اضافه می کنیم. برای آموزش از 4 کرنل، ضریب یادگیری 0.01 و 300 epoch استفاده می کنیم. بعد از اتمام یادگیری، با تابع predict، در بازه 3- تا 3 تست می کنیم و مقادیر به دست آمده را رسم می کنیم. در شکل زیر نتیجه مشاهده می شود:



ج) در سوال آخر تابع سینوس را در بازه 4- تا 4 رسم کرده ایم. همچنین مقدار حاصل از پیش بینی RBF و MLP را نیز رسم کرده ایم.