به نام خدا

ملیکا احمدی رنجبر 97521036

❖ سوال اول) KOHONEN

◆ **قسمت الف:** در این سوال ، پیاده سازی KOHONEN-SOFM از ما خواسته شده است ، ابتدا با هم توضیح مختصری در رابطه با این روش آموزش بدهیم و سپس به بررسی کد بپردازیم.

در این روش بر خلاف روشی که در سری گذشته تمرین داشتیم ، از روش Unsupervised استفاده میکنیم.به این معنا که در قبل ، ما جواب حقیقی را برای مساله داشتیم و ا توجه به جوا محاسبه شده خودمان و آن ، جواب مورد نظر مسئله را بهتر میکردیم. در اینجا ، به این صورت است که خود الگوریتم ما بر اساس داده ها باید بهترین Map را برای خروجی به ما بدهد ، به این شکل که ورودی های نزدیک به یکدیگر را از لحاظ Feature ر کنار و نزدیک یکدیگر قرار دهد. Map گفته شده ، صفحه نرون ما است و همانند قبل ما تعدادی ورودی و تعدادی وزن (بسته به ورودی و نرون) داریم. و نکته دیگر آن است که میتوان ورودی را با dimension بیشتر از دو ، بر روی Map 2D نمایش داد (یا حتی Map 1D).

اما مراحل انجام کار که کد بر اساس آن عمل میکند به چه شکل است؟

- در ابتدا باید مرحله Initialization انجام شود. یعنی با توجه به تعداد ورودی و Map (تعداد نرون) ، ماتریس وزن را Initialize میکنیم و مقادیری کوچک و رندوم به آن میدهیم.
- پس ار آن نوبت به مرحله انتخاب برترین میرسد. در این مرحله ما یکی از ورودی هایی را که به طور رندوم انتخاب کردیم را به این تابع میدهیم ، در جواب آن ، Min Euclidian Distance را پیدا میکنیم (اختلاف بین ورودی و وزن) و از Arc آن استفاده میکنیم. این مرحله ، مرحله رقابتی است و نرون ها سعی در برنده شدن دارند.
- پس از تمام شدن مرحله رقابت نوبت به مرحله همکاری بین نرون ها میرسد ، و نرون برنده بسته به شعاع همسایگی و تابعی که برای آن در نظر گرفته شده است ، نرون های نزدیک خودش را به خود نزدیک تر و نرون هایی که ارتباطی با آن ندارند را از خود دورتر میکند.
- در انتها مهمترین قسمت که Update Weights است را داریم و با توجه به فرمول همیشگی و همچنین همسایگی بدست آمده از مرحله پیش ، وزن را Update میکنیم.
- با تکرار تمام مراحل بالا به تعداد Iteration هایی که داریم ، میتوانیم به ترتیب Train شدن این Map و نرون ها را مشاهده کنیم و در انتها به پاسخ دلخواه برسیم.

برای مثال در همین سوال ابندا تمام رنگ ها پراکنده هستند اما با تعداد کمی Iteration میتوان Map را به زیبایی در انتها دید.

این سوال بدون در نظر گرفتن تغییرات Learning Rate و Radius بود.

◆ **قسمت ب:** توضیحات کلی این سوال همانند قسمت پیش است و تنها تفاوت آن تغییر دادن Learning Rate در طول مدت Training است. اگر این پارامتر در طول مدت یادگیری کاهش پیدا نکند ، همانند اتفاق پیش خواهد آمد که در مباحث گذشته داشتیم ، یعنی برای مثال در انتهای یادگیری که تا حد خوبی به جواب نزدیک شده ایم و نیاری به تغییرات زیاد نداریم Learning Rate با همان شدت گذشته در جواب ما تاثیر میگذارد و احتمال دارد بهترین جواب را به ما ندهد و مدام این طرف و آن طرف پاسخ خوب بیفتد ، پس بهتر است با گذشت زمان ، این مقدار بسته به هر نوع تابعی کاهش یابد. و در صورت ثابت بودن آن ممکن است به مقدار نهایی Converge نکند.

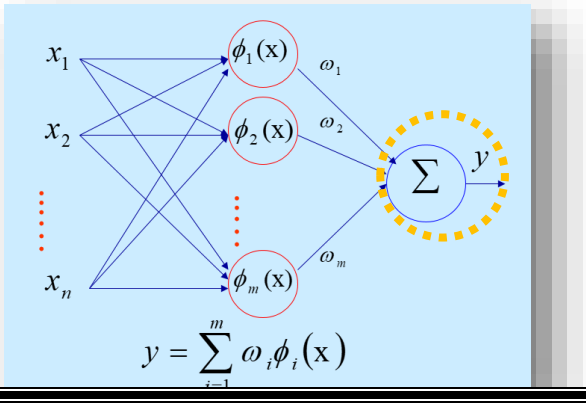
◆ **قسمت ج (امتیازی):** در این بخش مقاسه کاهش Radius را داریم. Radius برای انتخاب همسایگی و مقدار تاثیر گذاری آنها در Update Weightsاست و قطعا مهم است ، چراکه در صورت کم بودن آن ممکن است در دو طرف Map رنگ هایی موجود باشده که باید به هم نزدیک می بودند اما دور شدند و یکدیگر را جذب نکردند. اما اینکه ایم مقدار تا آخر یادگیری به بزرگی ابتدای آن باشد نیز ممکن است تاثیر منفی در نتیجه بگذارد و باعث تغییرات زیاد Map در صورتی که نیازی به تغییرات زیاد نیست شود.

❖ سوال دوم)

◆ **قسمت الف:** در این قسمت ، باید بر اساس یادگیری که در تمرین قبل داشتیم یعنی MLP تابع f(x) = sin(x) را آموزش دهیم و بتوانیم آن را پیشبینی کنیم. در قسمت بعد همین کار را با یکی دیگر از متد ها انجام دهیم و نتیجه این دو را محاسبه کنیم. در مورد MLP به طور کامل در تمرین گذشته بحث کردیم و دانستیم که یک لایه ورودی ، یک لایه خروجی و تعدادی لایه میانی (Hidden Layers) دارد. خروجی هر لایه به لایه بعدی میرود (Forward Propagation) و در انتها به محاسبه Error نهایی در لایه آخر ، این Error به شکل Back Propagation با لایه های پیشین باز میگردد و به همین شکل وزن ها را Update میکنیم.

◆ **قسمت ب:** در این روش هم ، برخی از مراحل مانند MLP است ، اما تفاوت های ساختاری آن ها نیز کم نیست. ابتدا توضیحی در مورد روش Radial Basis Function میبینیم و سپس روش را بررسی میکنیم. در این روش با استفاده از تغییر بعد یک ورودی سعی در دسته بندی کردن اطلاعات داریم ، در واقع همان کاری که در MLP توسط چندین لایه میانی انجام میشد ، حالا میتوان با استفاده از تعدادی Radial Basis Function انجام شود. ساختار کلی این روش دارای 3 لایه است ، لایه ورودی ، یک لایه میانی شامل تعدادی RBF و در نهایت هم لایه خروجی را داریم. لایه خروجی عملکرد بسیار ساده ای دارد و در واقع یک Perceptron است.

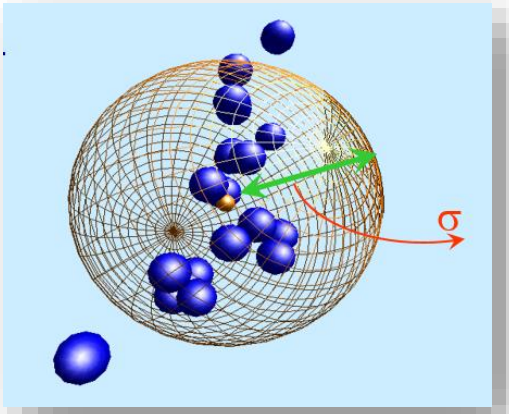
به شکل زیر توجه فرمایید :



اما هر کدام از این RBF ها چگونه عمل میکنند؟ میتوان توابع مختلفی برای این در نظر گرفت اما تابع مورد استفاده ما ، Gaussian است و به این شکل است:

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

که t مرکز RBF است و در مخرج هم ، شعاع را داریم.
همانطور که در تصویر میبینید. حال مراحل انجام این الگوریتم به این شکل است که:



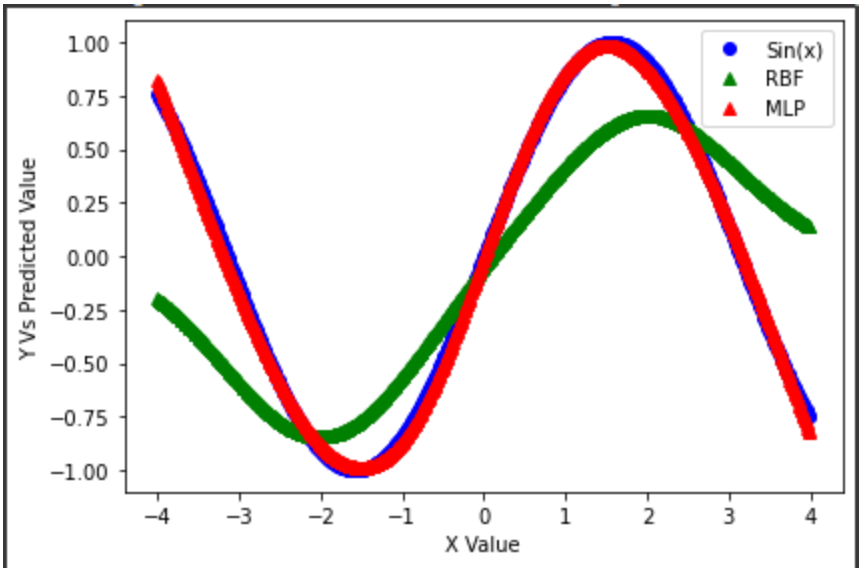
هر یک از Vector های ورودی را به تمام لایه میانی (یعنی تمام RBF ها) میدهیم و حاصل را محاسبه میکنیم. سپس به راحتی Error را با توجه به جواب اصلی برای آن ورودی خاص حساب میکنیم و همانند روش Stochastic Gradient که برای هر Input تابع Update Weights را صدا میزد ، ما هم وزن را Update میکنیم.
اما تنها نکته ای که این روش دارد ، در مورد تعداد این RBF ها است. با توجه به تعداد ورودی که داریم ، میتوانیم آنها را دسته بندی کنیم و به تعداد دسته بندی ها RBF در لایه میانی قرار دهیم ، در واقع تعدادی که ما برای RBF ها در نظر میگیریم ، تعداد دسته بندی ورودی هاست. یک تابع (KClustering) برای بدست آوردن این دسته بندی ها نوشته شده است و علاوه برا دسته بندی ، مراکز این دسته ها را نیز مشخص میکند و همچنین شعاع مناسب و خوب را نیز میتوان به کمک این تابع بدست آورد.
پس ما تمام فاکتور های لازم را داریم و Training را شروع میکنیم.
بعد از تمام شدن Training نوبت تست کردن شبکه ساخته شده و وزن های بدست آمده میرسد.

◆ **قسمت ج:** هر دو روش به طور کامل توضیح داده شد ، اما اکنون به سراغ تفاوت هایشان میرویم که در عکس زیر میتوان خلاصه آن را دید:

- Network structure
  - MLP : shares common neuron model
  - RBF : output and hidden layer different function
- Hidden node computation
  - RBF : *Euclidean norm*
  - MLP : *inner product*
- Approximation to non-linear input-output mapping
  - MLP : *Global*
  - RBF : *Local*
    - *fast, insensitive of order of sample data*
    - *need large data to get smooth surface*

RBF در ، Training سرعت بسیار بیشتری نسبت به MLP دارد.
تفاوت آموزششان را نیز میتوان در کد دید.

- **MLP**: uses dot products (between inputs and weights) and sigmoidal activation functions (or other monotonic functions such as ReLU) and training is usually done through backpropagation for all layers (which can be as many as you want). This type of neural network is used in deep learning with the help of many techniques (such as dropout or batch normalization);
- **RBF**: uses Euclidean distances (between inputs and weights, which can be viewed as centers) and (usually) Gaussian activation functions (which could be multivariate), which makes neurons more locally sensitive. Thus, RBF neurons have maximum activation when the center/weights are equal to the inputs (look at the image below). Due to this property, RBF neural networks are good for novelty detection (if each neuron is centered on a training example, inputs far away from all neurons constitute novel patterns) but not so good at extrapolation. Also, RBFs may use backpropagation for learning, or hybrid approaches with unsupervised learning in the hidden layer (they usually have just 1 hidden layer). Finally, RBFs make it easier to grow new neurons during training.



نتیجه نهایی هم به این شکل است :

MLP کمی دقیق تر از RBF عمل کرده است

شاید به دلیل تعداد RBF ها باشد.