به نام خدا

گزارش تمرین سری اول درس هوش محاسباتی

دکتر مزینی

محمد یارمقدم

96462104

سوال یک

در این سوال هدف طراحی یک perceptron **ساده بود. برای این منظور ابتدا کلاس موردنظر را برای این هدف ساخته و** constructor **آنرا نوشتم. در این بخش فیلد های** learning rate **، تعداد تکرار شبکه برای یادگیری ، وزن های اولیه ، بایاس و تابع فعالسازی مقداردهی شدند.**
وزن های اولیه و بایاس در ابتدا به صفر مقداردهی شدند. در قسمت بعدی تابع پله ایی را به عنوان تابع فعالسازی تعریف کرده و از کتابخانه numpy **استفاده تا به جای کسب نتیجه برای یک یک زوج تنها برای کل آرایه ورودی تابع کار کند.**

در قسمت بعدی توابع پیشبینی مقدار حاصل شبکه عصبی و fit **را نوشتم. در تابع** predict **طبق فرمول های پرسپترون مقادیر را محسابه کرده و مقدار حاصل از تابع فعالسازی را برگرداندم.**

طبق روابط داریم : $y = activation\_func(w.x + b)$

پس ابتدا با تابع دات از numpy **حاصل ضرب داخلی را محاسبه و سپس با بایاس جمع می کنیم.**

در نهایت خروجی تابع خطی را به تابع فعالسازی میدهیم تا مقدار 0 **یا** 1 **را به ما بدهد.**

در قسمت آخر هم تابع fit **را تکمیل کردم. در این تابع به تعداد** iteration **های لازم که به این تابع داده میشود و به تعداد سمپل های داده ورودی فرآیند آپدیت را طبق فرمول موردنظر انجام میدهیم.**

طبق رابطه داریم : $w := w + gradient \quad , \quad gradient = learning\_rate * (real\_y - predicted\_y)$

در قسمت آخر هم مین برنامه با مقدار دهی سمپل ورودی با تابع nor **انجام شد. در این قسمت یک** instance **از کلاس پرسپترون ساختم و روی آن توابع موردنظر را فراخوانی و مقدار وزن های بدست آمده و بایاس برای شبکه حاصل محاسبه شد.**

سوال دو

هدف در این سوال طراحی یک binary classification **استو برای این منظور از کلاس پیاده سازی شده در سوال قبل استفاده کردم.**

سپس در قسمت مین برنامه برای دریافت داده ورودی از تابع loadtxt **کتابخانه** numpy **استفاده کردم.**

البته در مرحله قبل لازم است که فایل data.txt **را در فولدر کولب در گوگل درایو آپلود کرده و از طریق** mount **کردن درایو در محیط کولب داده ، درایو را به محیط بشناسانیم و سپس از آدرس فایل داده ورودی را بخوانیم.**

در مرحله بعد داده تست و ورودی را تشکیل دادم و مانند موردقبل train **شبکه را انجام دادم.**

در مرحله آخر هم از طریق تابع scatter **از کتابخانه** matplotlib **نقاط داده ورودی را در صفحه رسم و در نهایت خط جدا کننده دو محیط را از طریق تابع** plot **انجام دادم.**

سوال سه

Madaline **در واقع ترکیبی از چند شبکه** Adaline **است. به این صورت خروجی شبکه ما به جای یک خروجی میتواند مجموعه از خروجی ها را داشته باشد تا بتواند بیش از دو کلاس را** classification **کند.**

از شبکه آدالاین برای جداکردن مرز ببین دوکلاس در داده های مختلف استفاده میشود ولی اگر از مادالاین استفاده کنیم میتوانیم به تعداد آدالاینی که در آن موجود است در دیتای ورودی مرز های بیشتری جدا کنیم و یک ناحیه مجزا ار داده را ایجاد کنیم.

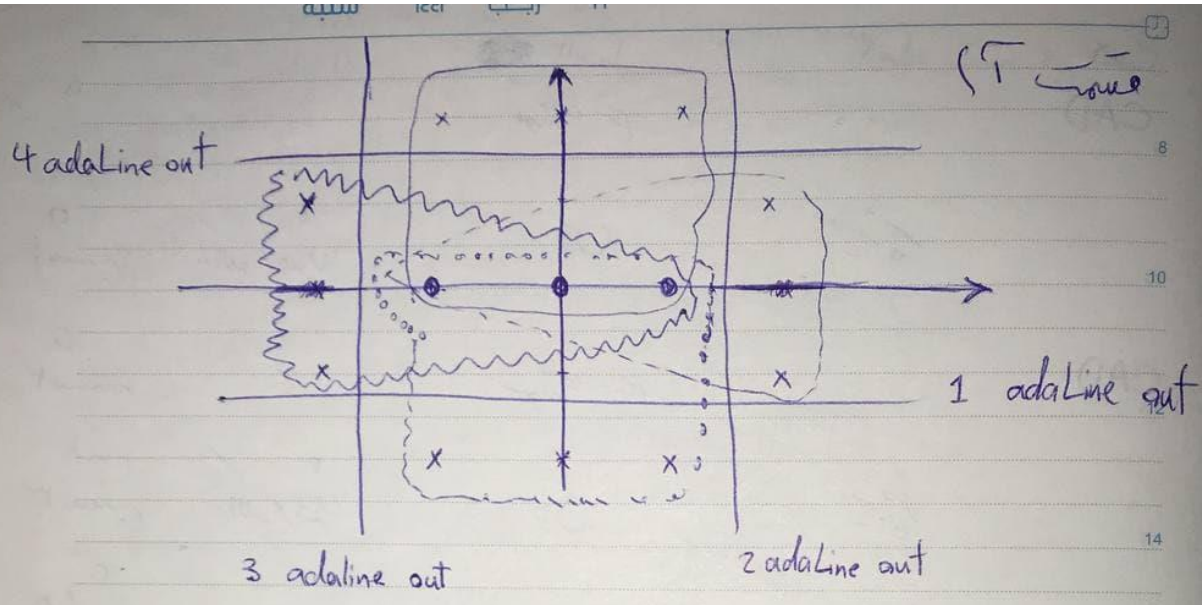
پس حتی اگر داده های ورودی غیر خطی نیز باشند چون شبکه مادالاین میتواند دسته بندی مجزا جدا کند و مابقی آدالاین ها هم دسته بندی دیگر کنند پس ناحیه غیرخطی را میتوانیم دسته بندی کنیم و داخل و خارج آنرا از طریق محمدود کردن ان به ناحیه بدست آوریم.

در قسمت دوم سوال مورد آ قابل انجام و مورد ب غیر قابل انجام است.

مورد ب به هیچ وجه قابل دسته بندی نیست. زیرا در هر صورت یکی از انواع داده در کلاس اشتباه قرار میگیرند.

شبکه موردنظر برای قسمت آ نیز به صورت زیر است:

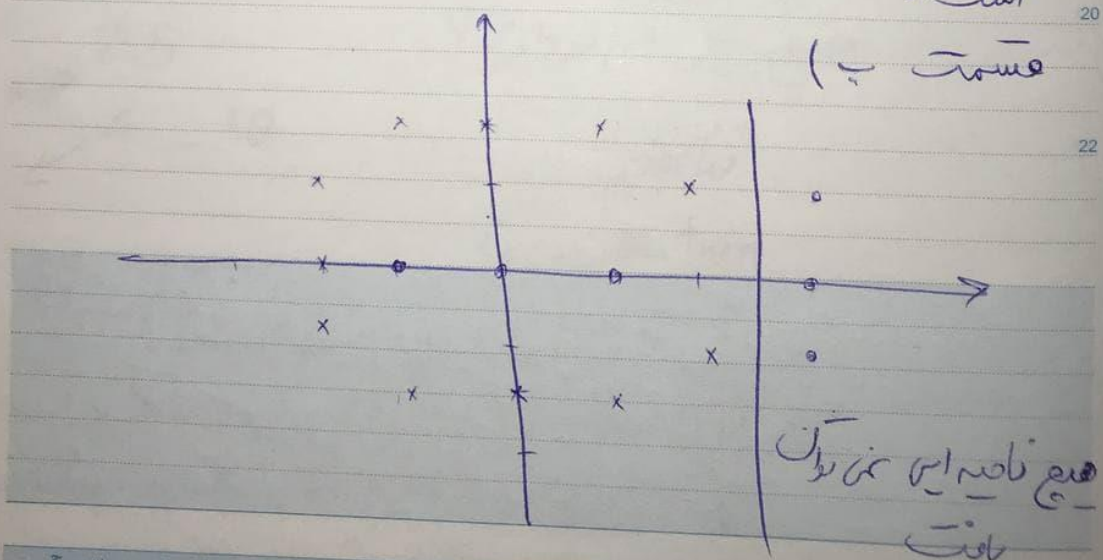
قسمت آ)



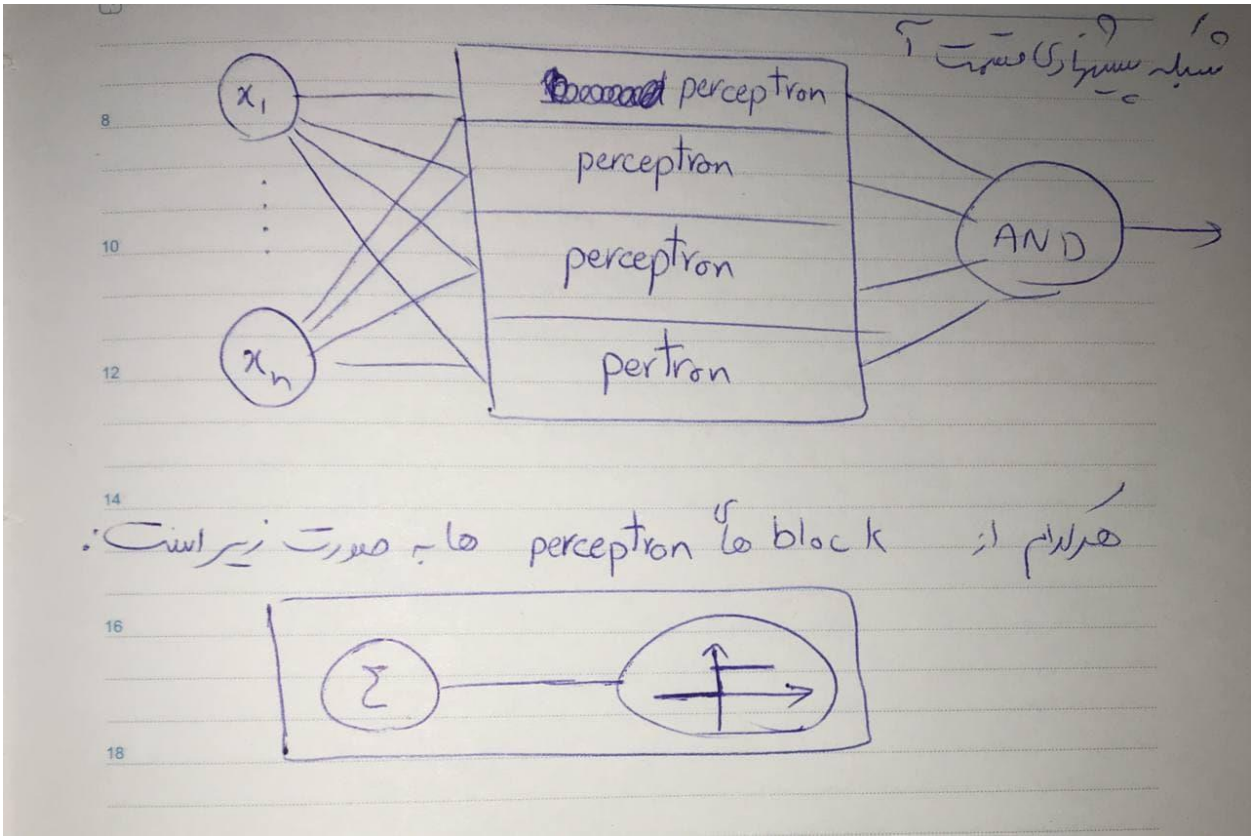
هرکدام از چهار خروجی بالا و وروری ها هرکدام از adline ها

خطوط ناحیه رسم شده برای جداسازی نواحی هر خروجی adeline است.

قسمت ب)



هیچ ناحیه ای نمی تواند جدا شود



هرکدام از perceptron ها به صورت زیر است:

سوال چهار

هدف در این سوال پیاده سازی multi layer perceptron **یا** MLP **است تا از طریق بتوانیم** classification **را بر روی دیتاست** mnist **انجام دهد.**

بدین منظور ابتدا از بین دیتاست های کتابخانه keras **دیتاست** mnist **را می خوانیم. این داده شامل دو دسته** (x_test, y_test) **و** (x_train, y_train) **است که در واقعا داده های ورودی نمونه و خروجی موردانتظار آن میباشد و دسته بعد شامل داده های تست میباشد . سپس باید تعداد لیبل ها مشخص شود که باید به تعداد داده های داده های ورودی باشد. در مرحله بعدی برای اینکه فرمت داده ورودی برای محاسبات راحت تر باشد از طریق تابع** to_categorical **هر لیبل را به فرمت** one hot **تبدیل کردم.**

در مرحله بعد تعداد لایه ورودی را محسابه کردم که از تعداد پیکسل های هر عکس در دیتاست محسابه شد.

مجددا برای سادگی مسیر از طریق تقسیم داده های تست و ترین بر 255 **مقادیر داده ورودی به جای** 0 **تا** 255 **به** 0 **تا** 1 **تبدیل کردم.**

از طریق تابع sequential **از کتابخانه** keras **مدل ابتدایی** MLP **را تشکیل دادم.**

سپس از طریق تابع add **لایه های نهان مورد نیاز خود را به این مدل اضافه کردم. که شامل لایه های موجود در عکس میباشد.**

در قسمت بعد مدل از با ویژگی های موردنیاز کامپایل میکنیم تا تشکیل نهایی آن صورت گیرد.

ویژگی ها آن شامل نوع optimizer **و نوع تابع** loss **است.**

سپس برای train **کردن مدل از تابع** fit **استفاده کردم که داده** trian **و خروجی نظیر آن ، تعداد** ecoch **ها و سایز** batch **را دریافت و مدل را آموزش میدهد. در قسمت آخر هم** loss **و** accuracy **هر** epoch **محاسبه و توسط تابع** plot **نمودار های موردنظر رسم شد.**

سوال پنج

در این سوال هدف پیاده سازی MLP **توسط** numpy **است که توسط آن بتوان بر روی دیتاست** mnist ، classification **انجام داد.**

بدین منظور همانند سوال چهار ابتدا دیتاست را خوانده و داده های آن را به 0 **تا** 1 **نگاشت میکنیم. سپس داده های** train **را از** train_test_split **میخوانیم.**

در قسمت بعد کلاس MLP **را تشکیل دادم که در اینجا** DeepNeuralNetwork **نامگذاری شده است.**

در این کلاس برخی از توابع ساده هستند و پیاده سازی سنگینی ندارند همانند sigmoid **و** softmax **که مشخص هستند.**

در قسمت کانستراکتور برنامه نیز مقادیر sizes **،** epochs **،** Learning_rate **و** params **مقداردهی شدند.**

Sizes **در واقع یک آرایه است که اندازه لایه های شبکه را در خود جای میدهد.** Params **نیز یک دیکشنری از تمامی پارامترهای شبکه است.**

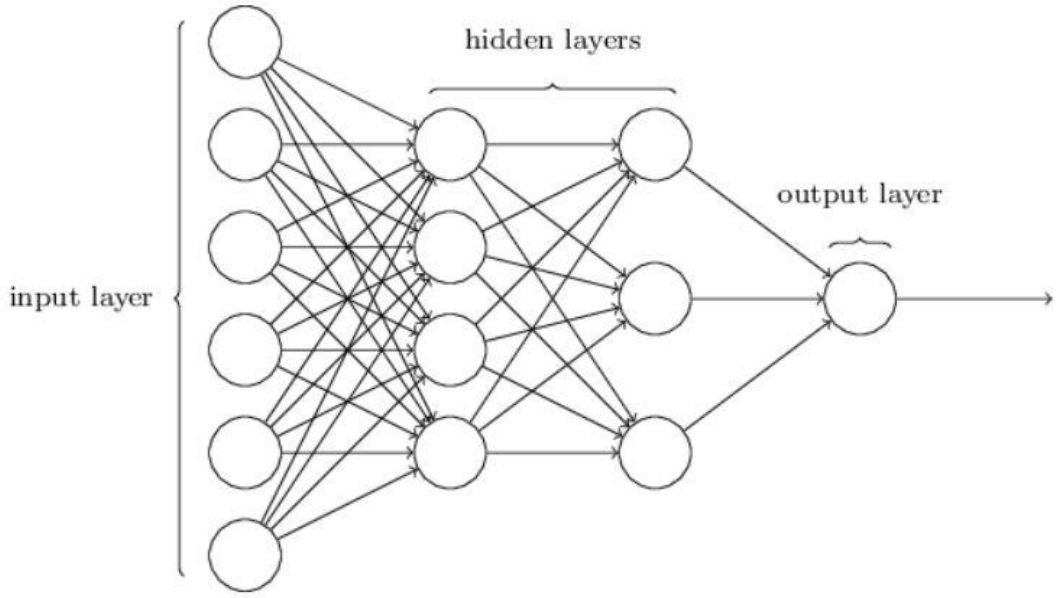
تابع بعدی initialization **نام دارد که توسط سایز های موجود در آرایه** size **لایه ها را میسازد. همچنین در** params **از طریق تابع رندوم آرایه های لایه های شبکه مقداردهی رندوم میشود و لایه ها تشکیل میشوند.**

تابع بعدی forward_pass **است که در آن از داده های** trian **آغاز میکنیم و مقادیر مختلف لایه های اول و دوم و سوم را به ترتیب آپدیت میکنیم که این کار از ضرب داخلی وزن لایه در داده ورودی لایه محاسبه میشود که به تابه سیگموید داده میشود.که داده ورودی هر لایه در واقعا خروجی لایه قبل آن است. در نهایت وزن خروجی لایه آخر برگشت داده میشود.**

تابع بعدی backward_pass **است که دقیقا خلاف تابع قبل است که در آن باید از خروجی داده آخر و از طریق مشتق گیری که فرمول های آن در عکس موجود است میزان آپدیت های وزن های لایه ها را برای کم کردن** loss **محاسبه کنیم.**

تابع compute_accuracy **نیز دقت** train **را محسابه میکند.**

اصلی ترین تابع این کد تابع train **است که به تعداد** iteration **هایی که تعریف میشود در کل داده های** train **ابتدا خروجی را مجاسبه میکند (**forward_pass**) و سپس تغییر وزن ها محاسبه میشود(**backward_pass**) و در آخر از طریق تابع** update_network **وزن ها آپدیت میشوند.**



ساختار شبکه استفاده شده

$$MSE = \frac{1}{m}\sum_{i}^{m}(y - a^{L})^{2}$$

$$\frac{\partial Loss}{\partial w^L} = \frac{\partial Loss}{\partial a^L}\frac{\partial a^L}{\partial z^L}\frac{\partial z^L}{\partial w^L}$$

$$\frac{\partial Loss}{\partial w^L} = (y - a^L)\sigma'(z^L)a^{L-1}$$

$$\frac{\partial Loss}{\partial b^L} = \frac{\partial Loss}{\partial a^L}\frac{\partial a^L}{\partial z^L}\frac{\partial z^L}{\partial b^L}$$

$$\frac{\partial Loss}{\partial b^L} = (y - a^L)\sigma'(z^L)$$

$$\frac{\partial Loss}{\partial a^{L-1}} = \frac{\partial Loss}{\partial a^L}\frac{\partial a^L}{\partial z^L}\frac{\partial z^L}{\partial a^{L-1}}$$

$$\frac{\partial Loss}{\partial a^{L-1}} = (y - a^L)\sigma'(z^L)w^L$$

$$\frac{\partial Loss}{\partial w^{L-1}} = \frac{\partial Loss}{\partial a^{L-1}}\sigma'(z^{L-1})a^{L-2}$$

$$\frac{\partial Loss}{\partial b^{L-1}} = \frac{\partial Loss}{\partial a^{L-1}}\sigma'(z^{L-1})$$

$$\frac{\partial Loss}{\partial w_L} = a_{L-1}^T(y - a_L)\sigma'(z_L)$$

$$\frac{\partial Loss}{\partial b_L} = (y - a_L)\sigma'(z_L)$$

$$\frac{\partial Loss}{\partial a_{L-1}} = (y - a_L)\sigma'(z_L)w_L^T$$

در محاسبه و چگونگی استفاده از روابط از لینک زیر کمک گرفته شد:

https://github.com/Huggah/MNIST-Classification-with-Numpy-and-Backpropagation/blob/master/README.md