بسم الله الرحمن الرحیم

محمد عرفان زارع زردینی

سری ۱ تمرین هوش مصنوعی

سوال۱)

(رفرنس ها:

https://qa.deeplearning.ir/1115/%D8%AA%D9%81%D8%A7%D9%88%D8%AA-%D8%A8%DB%8C%D9%86-%D8%A7%D9%84%DA%AF%D9%88%D8%B1%DB%8C%D8%AA%D9%85-%D9%87%D8%A7%DB%8C-%D8%A8%D9%87%DB%8C%D9%86%D9%87-%D8%B3%D8%A7%D8%B2%DB%8C-gd-%D8%A8%D8%A7-batch-gd-%D9%88-mini-batch-gd-%D9%88-sgd-%DA%86%DB%8C%D8%B3%D8%AA%D8%9F

https://www.baeldung.com/cs/gradient-stochastic-and-mini-batch

https://medium.datadriveninvestor.com/batch-vs-mini-batch-vs-stochastic-gradient-descent-with-code-examples-cd8232174e14

https://www.geeksforgeeks.org/difference-between-batch-gradient-descent-and-stochastic-gradient-/descent

(

در SGD(Stochastic Gradient Descent) ، یک تقریبی از batch یا همان GD (standard gradiend descent) است که می آید یک نمونه تصادفی را انتخاب میکند و محاسبات روی آن انجام و تغییرات روی آن اعمال می شود که به شکل online انجام می شود و سریعتر از batch است وبرخلاف آن داده های کمتری برای دستکاری پس از زمانی وجود دارد.این روش نسبت به batch سریعتر و کم هزینه تر است ومحاسباتش سبکتر است.برای داده های آموزشی بزرگ هم کاربرد دارد.این روش راه حل خوبی ارائه میدهد ولی بهینه نیست ولی همگرایی سریعتر نسبت به batch رخ میدهد.

در حالی که در batch، نسبت به کل دیتاست حساب می شود وچون گرادیان تمام trainig set ها برای هر آپدیت محاسبه شود، سبب ایجاد سربار خیلی زیادی میشود(کل دیتا ها خونده و گرادیان ها محاسبه ومیانگین گرفته می شود وسپس پارامترهامون اپدیت می شوند.درواقع برای هر اپدیت کل دیتاست خوانده می شود) ودر صورت بزرگ بودن دیتاست ها ، سبب می شود این روش عملی نباشد و در مموری جا نشود.این روش کند وهزینه زیادی هم دارد وبرای داده های آموزشی بزرگ همانطور که گفتیم خوب نیست.این روش راه حل بهینه را با توجه بهزمان کافی،برای همگرایی ارائه می دهد اما همگرایی به کندی رخ میدهد.

Mini batch در واقع میان این دوتا هست و برخی از ویژگی های هرکدام را داراست. در این روش گرادیان به ازای چند نمونه محاسبه می شود(در واقع انگار دسته بندی روی دیتاها انجام و چند نونه بررسی می شوند)(گرادیان محاسبه و وزن ها آپدیت و بچ جدید با دیتای جدید می رسد.) که سبب می شود گرادیان ایجادی، پایدار ترو پیاده سازی برداری آن برایمان راحت تر می باشد.این روش سبب می شود زودتر به حداقل تابع هزینه برسیم. این روش با گرفتن زیر مجموعه ای از داده ها، تکرار کمتری نسبت بهSGD دارد وحجم محاسباتی کمتری نسبت به GD(BATCH) دارا است.این روش معمولا به دو روش بالا در اموزش ماشین ترجیح داده می شود.

اگر بخواهیم به شکل مرتب شده و جدا از نکات و مقایسه هایی که بالا انجام شد(به ویژه برای batch و SGD)جدولی برای خوبی ها و بدی های دو روش SGD وBATCH بنویسیم داریم:

روش	خوبی	بدی
SGD (Stochastic Gradient Descent)	1.سریعتر هست 2.محاسباتش سبکتره 3.تصادفی سازی، سبب generalization مدل می شود	1. جواب حاصل،دارای noise بیشتری نسبت به روش مشابه هست 2. سببoverfit می شود که واریانس بزرگ وbias کوچک میشود.
GD(batch) (Gradient Descent)	1.نویز کم تری دارد و ارور آن کمتر خطا دارد. 2.تخمین گرادیان آن unbiased هست	1.زمانبر هست 2.هزینه محاسبات آن زیاد است

روش SGD همانطور که گفتیم به علت این که روی یک نمونه تنها بررسی انجام میدهد، دارای نویز زیادی هست(چون ما دونه به دونه برای هر نمونه محاسبه میکینم و وزن ها رو اپدیت میکنیم و هر بار داریم حالت شک و تردید در خروجی رو بیشتر می کنیم.همچنین خیلی از نمونه ها دارای داده های پرت می باشد و ما داریم روی همین دیتا ها خودمون رو بروز می کنیم.) و خروجی آن واریانس بالا و bias کوچکی دارد همچنین روش بهینه ای نیست و دقت کمتری دارد.

با استفاده از روش momentum که در پایین هم بیان شده ،سبب می شود که پارامتر تتا،خیلی تغییر زیادی نکند ونوسان های شدیدی نداشته باشیم. در واقع با بهره از آن گویی هنگامی که در یک جهت حرکت میکنیم و وقتی از درستی مسیر اطمینان حاصل شد، گام هایمان رو بزرگتر میکنیم و در همان جهت ادامه می دهیم. تا به هدف وجواب نزدیکتر شویم.وقتی به جواب نزدیکتر شدیم ،گام هایمان کوچکتر می شود. در این روش اگر جهت در خلاف جهت درست باشد، حرکت لغو دوباره بررسی میشود.

$$\Delta\theta = \eta\Delta\theta - \alpha\nabla\mathcal{J}(\theta)$$

$$\theta = \theta + \Delta\theta$$

که از ترکیب این دو فرمول به فرمول روبرو می رسیم.

$$\theta = \theta - \alpha \nabla \mathcal{J}_i(\theta) + \eta \Delta \theta$$

سوال۲)

الف)

چون هر نقطه مدنظر(کلیدی) دارای مختصات xوy می باشد، به 10 نورون برای یافتن آن ها نیاز داریم. پس تعداد نورون لایه اخر 10تا می باشد.

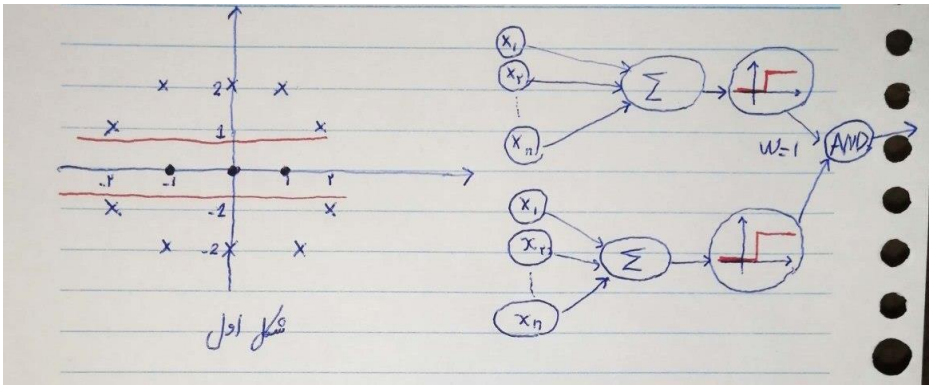
برای تابع ضرر ازmse استفاده می شود تا از پیش بینی پرت جلوگیری شود. تابع فعالساز(activition function) این سوال از sigmoid می باشد چون عدد میان صفر تا یک باشد تا مشخص کند نقطه مورد بررسی مهم هست یا نه.

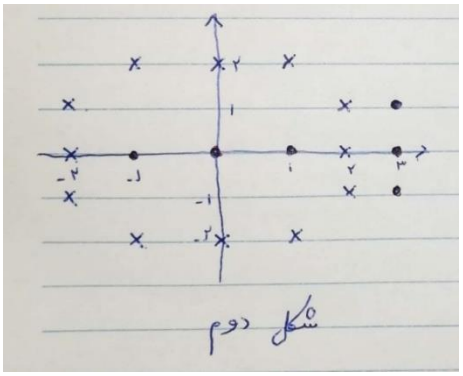
ب)

در کد داده شده از تابع ضرر mse with don’t care و تابع فعالساز sigmoid استفاده کردیم. تابع ضرر اشاره شده، فاصله اقلیدسی میان نقاط پیش بینی شده مان با نقاط درست و واقعی را محاسبه می نماید. نقاطی که توسط ما قابل پیدا شدن نیست ، در مجموعه با فرمت(0و0) نمایش داده می شود. البته این نقاط در محاسبه تابع ضرر دخالتی ندارند و حذف می شود.

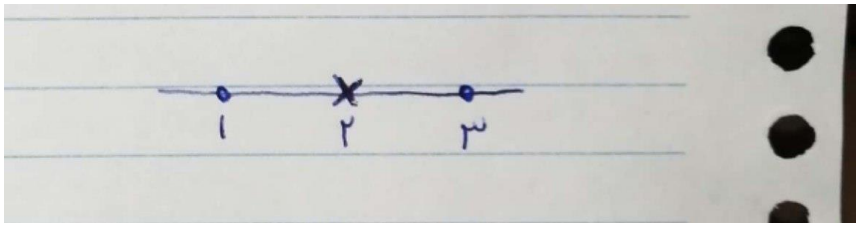
سوال۳)

Madalines شامل چند adalines می باشد.پس به جای یک خروجی ،چند خروجی با ماتریس وزن ها داریم. پس linear classification آن AND شدند با هم، پس این ویژگی سبب می شود که ما محدوده هایی با خطوط کشیده شده بسازیم. از آنجایی که از madalines میتوان برای classify کردن داده در چند ضلعی های محدب استفاده کرد، تصویر اول با دوتا adalines به کلاس های مناسب دسته بندی می شود.از آنجایی که تنها یک ناحیه محدب وجود دارد، هیچ لایه پنهانی موجود نیست.





درتصویر دوم اما به دلیل وضعیتی که در شکل زیر هم بدان پرداخته شده، شکل محدبی وجود ندارد که بتوان دو کلاس مشخص شده رو از هم جدا کرد.(شکل زیر برشی از شکل دوم هست)



سوال۴)

الف)

در مقام مقایسه perceptron ، برای اپدیت وزن ها و error ها ، از binary response که classification نتیجه می دهد استفاده می کند.در حالیکه adadline از continous response استفاده می کند که سبب می شود خطا ها بیشتر به مقدار واقعی نزدیک شود. به همین سبب قابلیت تعمیم adaline بیشتر از perceptron می باشد. الگوریتم madaline نسبت به adaline لایه های بیشتری دارد ، در نتیجه قابلیت تعمیم بیشتری دارد.MLP چون از Activion Function های غیرخطی نیز می تواند استفاده کند، پس برای classificationهای پیچیده تر هم قابل استفاده است و درنتیجه قابلیت تعمیم بیشتری نسبت به بقیه دارد.

ب)

Overfit زمانی رخ می هد که یک model روی training data خوب عملکرده ولی روی test data و داده های جدید ورودی عملکرد مناسبی ندارد. دراین حالت model جزئیات ونویز های training data را می اموزد که سبب اثر منفی روی شبکه موجود هنگام اجرا و بررسی داده های جدید می گذارد که علل آن می تواند شامل تمیز نبودن ودارای نویز بودن داده استفاده شده برای training، دارا بودن واریانس بالای model ،کافی نبودن سایز training dataset و یا پیچیده بودن زیاد model از لحاظ دارا بودن لایه زیاد و یا نورون های زیاد در هر لایه.

ج)

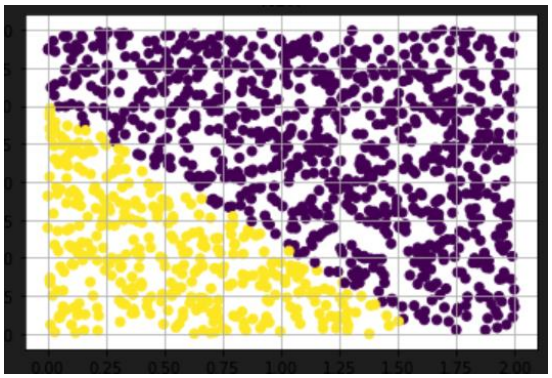
بزرگتر کردن training dataset_استفاده از regularization ها مانند weight decay _ساده کردن model_توقف training در زمان کوتاه تر

سوال۵)

همانطور که می دانیم ،nor ساختارش به شکل زیر می شود و دارای 2ورودی مان ، bias و خروجی عبارت nor شده مان هست.من برای سه تاsample به شکلrandom تولید شد بررسی کردم تا روش والگوریتم و ساختار پیدا شود سپس برای 1200 نقطه رندوم بررسی انجام شد.

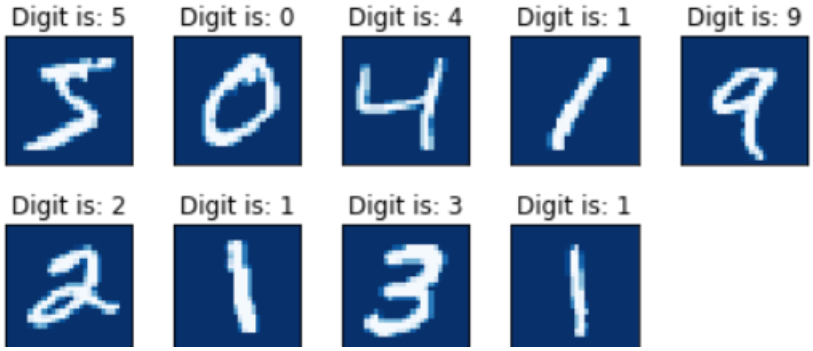
```
dataset=[[0,0,1,1],[0,1,1,0],[1,0,1,0],[1,1,1,0]]
```

روش مدنظر همان batch می باشد که ساختار و فرمت آن به شکل کامل در سوال اول بسط و توضیح داده شده است . مقدار learning rate رو من عدد 0.1 قرار دادم تا بررسی و در فرمول اعمال شود .همچنین حداکثر تغییر یا در واقع همان گام تغییرمون رو ابتدا 1 در نظر گرفتم و در حلقه زده شده گفتم از مقداری وقتی تغییرش بعد از اپدیت کمتر بود و در واقع گامش کمتر بود، بیاد و از بررسی دست بکشد وگرنه بیاد گرادیان را از فرمولی که دارد محاسبه کند و بر آن اساس وزن ها را اپدیت کند. مقادیر وزن اولیه را هم ابتدا در یک متغیر دیگر نگه می دارم تا تغییرات آن را بسنجم . سپس می ایم در کد مقدار گام رو اپدیت می کنم تا زمانی که شرط حلقه که در بالا اشاره شد، نقض شود. حال مثال ها را بررسی و بر اساس دیتا های دیتاست تصمیم گیری می کنم و در جایی ذخیره کرده و در نهایت با وزن های نهایی که داریم وتولید تعدادی نقطه رندوم و بررسی آن ها به روش گفته شده، نقاط مطلوب را چاپ کردم. و در نمودار نمایش دادم. تصاویر در فایل موجود هست ولی من اسکرین شات های خروجی رو هم در اینجا قرار می دهم.



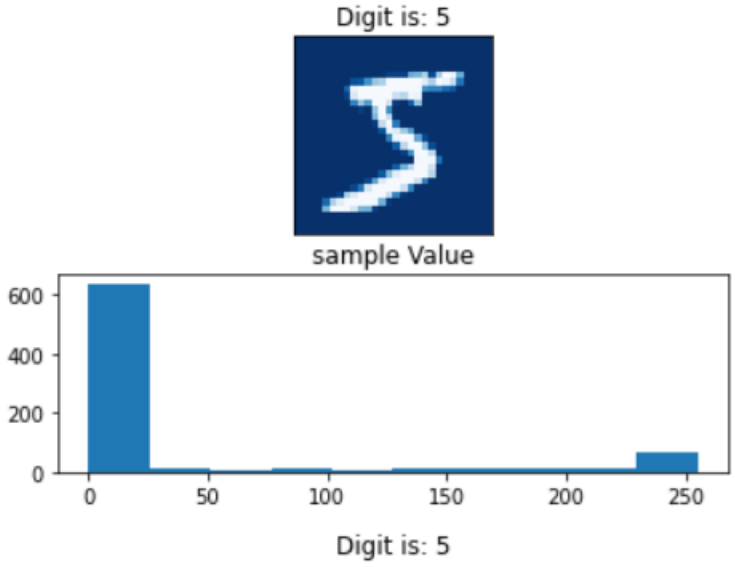
سوال۶)

ابتدا کتابخانه ها و دیتاست هایی که لازم هست برای حل سوال رو اضافه میکنیم.سپس داده ها زا از دیتاست فرا میتوانیم تا داده های trainو test مشخص شوند. سپس 9 مثال (میتواند بیشتر هم باشد ولی برای سنجیدن نامیش و تعداد معقول بودن 9گذاشتم) را از داده های train شده پردازش و شکل و عدد نظیر آن را نمایش می دهیم.



سپس می آییم داده اولtrain مون رو برای نمونه بررسی میکنیم و ساختار آن و داده های نمونه را در نمودار نمایش میدهیم.

سپس داده ها رو دوباره reshape کرده و نوع و سایزشون رو مشخص میکنیم وسپس از طریق داده هارو encodeمی کنیم وبرای نمایش این که پس از ان تغییرش چگونه هست چاپ می کنیم .




```
shape train matrix (60000, 784)
shape Test matrix (10000, 784)
(array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9], dtype=uint8), array([5923, 6742, 5958, 6131, 5842, 5421, 5918, 6265, 5851, 5949]))
Shape before encode:  (60000,)
Shape after encode:  (60000, 10)
after encode [0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
```

حال با مشخص کردن تابع فعالساز و اینکه دستورdropout زده می شود تا برخی از نورو نهای انتخابی در طول تمرین نادیده گرفته شده و سهم آن ها در فعالسازی نورون پایین دست در گذر زمان ، از بین میرود. وهرگونه اپدیت وزن نورون در backward نادیده گرفته می شود.سپس ان را با مشخص کردن optimizer مد نظر وloss ان کامپایل ودر نهایت با مشخص کردن سایز batch وداده های valid ، نمودارهای مدنظر را رسم میکنیم.(من برای دوره زمانی 20 در نظر گرفتم که میشد با دوره زمانی کمتر هم نمایش داد ولی دقیق تر هست.)

```
Epoch 1/20
469/469 - 6s - loss: 0.2691 - acc: 0.9209 - val_loss: 0.1104 - val_acc: 0.9658 - 6s/epoch - 12ms/step
Epoch 2/20
469/469 - 5s - loss: 0.1047 - acc: 0.9679 - val_loss: 0.0830 - val_acc: 0.9751 - 5s/epoch - 10ms/step
Epoch 3/20
469/469 - 5s - loss: 0.0740 - acc: 0.9776 - val_loss: 0.0665 - val_acc: 0.9797 - 5s/epoch - 10ms/step
Epoch 4/20
469/469 - 5s - loss: 0.0542 - acc: 0.9830 - val_loss: 0.0732 - val_acc: 0.9793 - 5s/epoch - 10ms/step
Epoch 5/20
469/469 - 5s - loss: 0.0442 - acc: 0.9857 - val_loss: 0.0660 - val_acc: 0.9802 - 5s/epoch - 10ms/step
Epoch 6/20
469/469 - 5s - loss: 0.0351 - acc: 0.9884 - val_loss: 0.0649 - val_acc: 0.9798 - 5s/epoch - 10ms/step
Epoch 7/20
469/469 - 5s - loss: 0.0313 - acc: 0.9898 - val_loss: 0.0656 - val_acc: 0.9816 - 5s/epoch - 10ms/step
Epoch 8/20
469/469 - 5s - loss: 0.0273 - acc: 0.9911 - val_loss: 0.0751 - val_acc: 0.9809 - 5s/epoch - 10ms/step
Epoch 9/20
469/469 - 5s - loss: 0.0257 - acc: 0.9912 - val_loss: 0.0708 - val_acc: 0.9814 - 5s/epoch - 10ms/step
Epoch 10/20
469/469 - 5s - loss: 0.0243 - acc: 0.9920 - val_loss: 0.0749 - val_acc: 0.9809 - 5s/epoch - 10ms/step
Epoch 11/20
469/469 - 5s - loss: 0.0213 - acc: 0.9929 - val_loss: 0.0665 - val_acc: 0.9821 - 5s/epoch - 10ms/step
Epoch 12/20
469/469 - 5s - loss: 0.0186 - acc: 0.9939 - val_loss: 0.0648 - val_acc: 0.9832 - 5s/epoch - 11ms/step
Epoch 13/20
469/469 - 5s - loss: 0.0189 - acc: 0.9935 - val_loss: 0.0638 - val_acc: 0.9833 - 5s/epoch - 10ms/step
Epoch 14/20
469/469 - 5s - loss: 0.0159 - acc: 0.9942 - val_loss: 0.0815 - val_acc: 0.9815 - 5s/epoch - 10ms/step
Epoch 15/20
469/469 - 5s - loss: 0.0165 - acc: 0.9943 - val_loss: 0.0836 - val_acc: 0.9814 - 5s/epoch - 10ms/step
Epoch 16/20
469/469 - 5s - loss: 0.0153 - acc: 0.9949 - val_loss: 0.0869 - val_acc: 0.9804 - 5s/epoch - 10ms/step
Epoch 17/20
469/469 - 5s - loss: 0.0134 - acc: 0.9956 - val_loss: 0.0776 - val_acc: 0.9828 - 5s/epoch - 10ms/step
Epoch 18/20
469/469 - 5s - loss: 0.0125 - acc: 0.9958 - val_loss: 0.0763 - val_acc: 0.9829 - 5s/epoch - 10ms/step
Epoch 19/20
469/469 - 5s - loss: 0.0137 - acc: 0.9955 - val_loss: 0.0762 - val_acc: 0.9834 - 5s/epoch - 10ms/step
Epoch 20/20
469/469 - 5s - loss: 0.0116 - acc: 0.9962 - val_loss: 0.0894 - val_acc: 0.9825 - 5s/epoch - 10ms/step
```