بسمه تعالی

دانشکده مهندسی کامپیوتر

آبان ۱۴۰۰

مبانی هوش محاسباتی

نام استاد: دکتر مزینی

تمرین اول

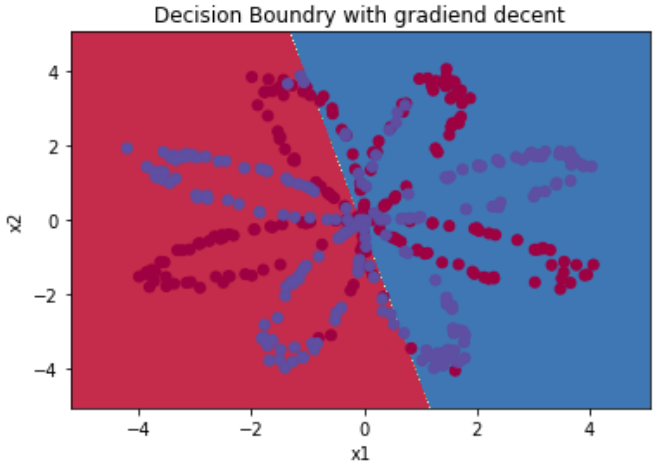
آرمان حیدری

شماره دانشجویی: ۹۷۵۲۱۲۵۲

۱. پاسخ سوال اول

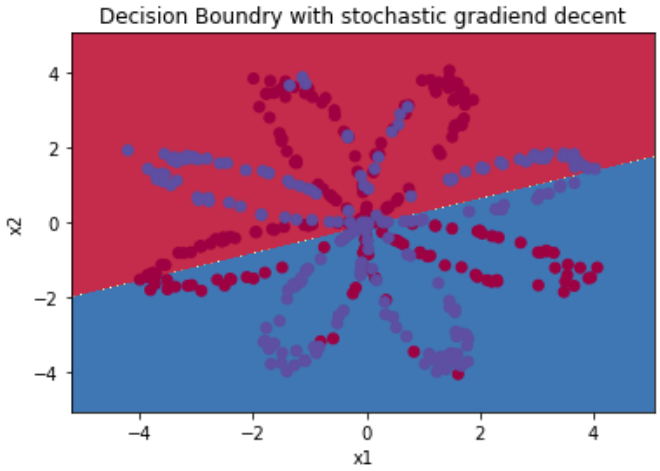
اگر متغیر is_stochastic در hyperparameterهای شبکه، True باشد، سوال با SGD حل میشود. و در غیر این صورت gradient decent معمولی خواهد بود.

در حالت gradient decent:




```
loss in epoch 90 : 0.1415559130749333
loss in epoch 91 : 0.14069248482997462
loss in epoch 92 : 0.13984893922353311
loss in epoch 93 : 0.13902603886958612
loss in epoch 94 : 0.13822446038035788
loss in epoch 95 : 0.1374447907500093
loss in epoch 96 : 0.13668752485353564
loss in epoch 97 : 0.13595306407866162
loss in epoch 98 : 0.13524171608104965
loss in epoch 99 : 0.13455369562695918
loss in epoch 100 : 0.133889126463585
train accuracy: 47.25 %
```

در حالت stochastic gradient decent: (batch_size=16)




```
loss in epoch 990 : 0.11172001394249054
loss in epoch 991 : 0.11025387583128801
loss in epoch 992 : 0.13016728209537515
loss in epoch 993 : 0.12107502579399848
loss in epoch 994 : 0.13876486601952867
loss in epoch 995 : 0.12982225683886373
loss in epoch 996 : 0.12279777764768011
loss in epoch 997 : 0.12542818799968505
loss in epoch 998 : 0.10452756247876017
loss in epoch 999 : 0.12147649656604903
loss in epoch 1000 : 0.11454575033530925
train accuracy: 48.0 %
```

در حالت اول حرکت ما به سمت نقطه بهینه کاملا بدون نویز است. چون هر بار تمام داده ها را حساب میکنیم. اما در حالت دوم مقادیر loss نویز دارند و کمی نوسان میکنند، اما به طور کلی کاهش پیدا میکند و جهت اصلی الگوریتم صحیح است. با استفاده از الگوریتم SGD مشکل گیر کردن GD در مینیمم های محلی به نوعی حل میشود، چون بعید است که مینیمم محلی وجود داشته باید که برای هر batch مینیمم باشد.

مزیت مهمی که SGD دارد این است که استفاده از مموری کمتر است. ما در هر epoch فقط به اندازه batch size باید داده ها را بررسی کنیم و این هم در زمان انجام هر epoch و هم در مموری ما صرفه جویی میکند.

از لحاظ دقت به دست آمده نمیتوان یکی را بر دیگری برتر دانست چون با چندین بار اجرا میبینیم که هردو الگوریتم بین ۴۷ تا ۵۷ درصد نوسان میکنند.

این نکته که خط جدا کننده در دو روش متفاوت به دست آمده هم ربط به وزن های اولیه دارد که مقادیر رندومی بوده اند.

نکته: برای مقایسه بهتر در حالت SGD، epoch=1000 و در حالت GD، epoch=100 میگذاریم. چون در SGD درواقع هر بار فقط ۱۶ داده را بررسی میکنیم.

بهینه ساز هایی بهتر از این دو روش هم وجود دارد. مشکل مهمی که این دو روش دارند، ثابت ماندن نرخ آموزش است. اگر نرخ آموزش با جلو رفتن الگوریتم افزایش بیابد یا از نرخ آموزش دومی مانند momentum استفاده شود میتواند نتیجه را بسیار بهتر کند. البته با دیتاست های واقعی و بزرگ امروزه، استفاده از GD به علت طولانی بودن هر epoch و همچنین مموری زیادی که در حین اجرا اشغال میکند دیگر مرسوم نیست.

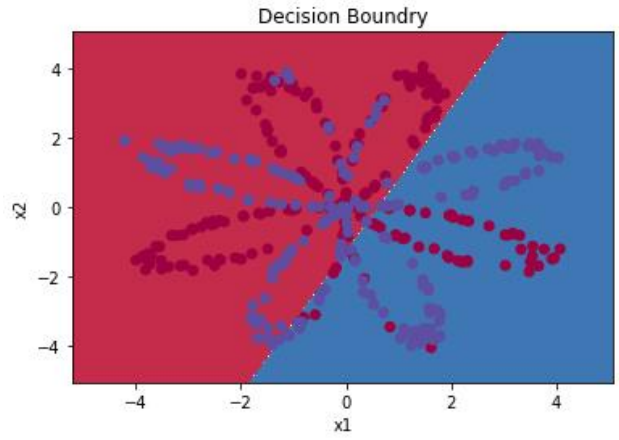
۲. پاسخ سوال دوم

پس از پیاده سازی به قسمت های زیر پاسخ میدهیم:

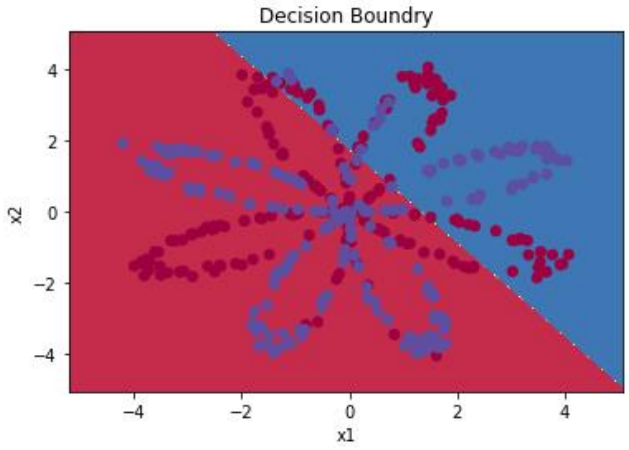
تاثیر تعداد نورون در شبکه با یک لایه مخفی) با ثابت نگه داشتن تمام هایپرپارامترهای شبکه با مقادیر زیر در هر حالت دقت را امتحان میکنیم:

```
learning_rate = 0.01
epochs = 2000
is_stochastic = True
batch_size = 32
threshold = 0.5
```

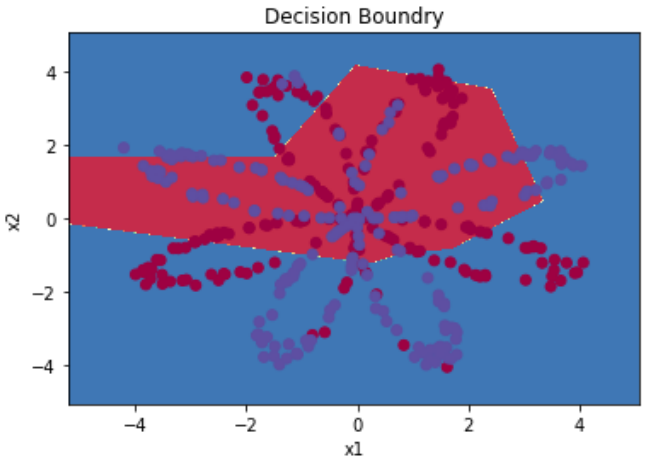
ابتدا hidden_layers=[1]: (دقت ۵۹.۲۵ درصد)



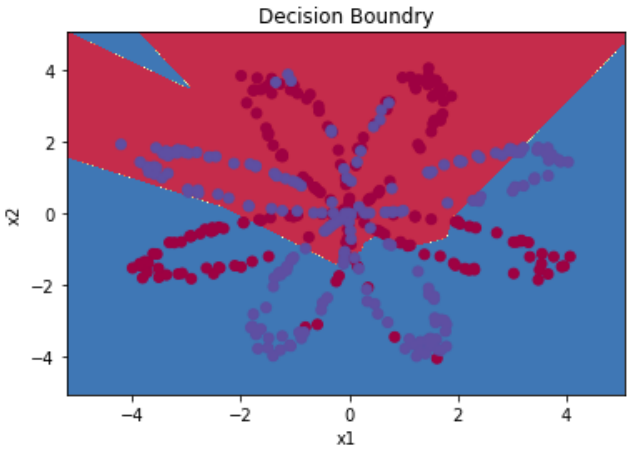
hidden_layers=[2]: (دقت ۴۴.۵ درصد)



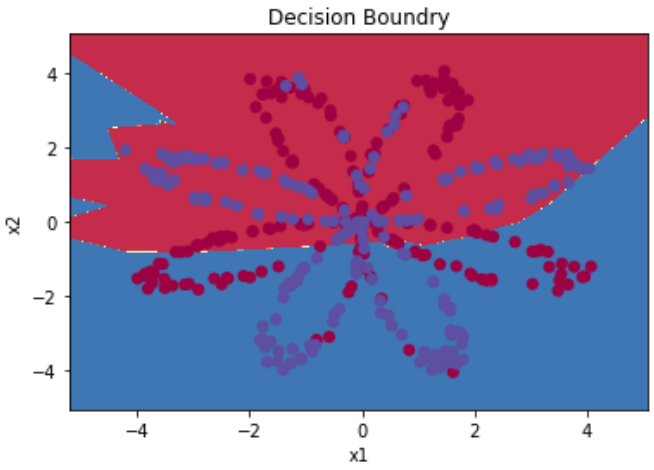
hidden_layers=[5]: (دقت ۵۲.۵ درصد)



hidden_layers=[20]: (دقت ۵۲.۷ درصد)



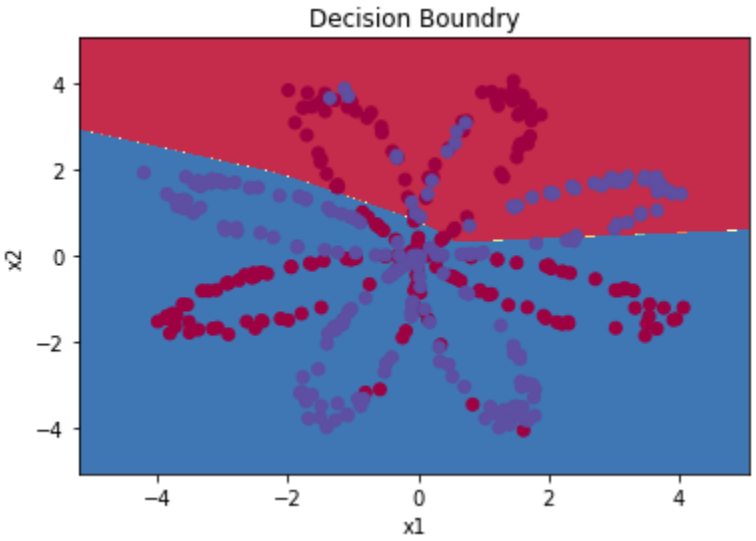
hidden_layers=[50]: (دقت ۵۱.۷ درصد)



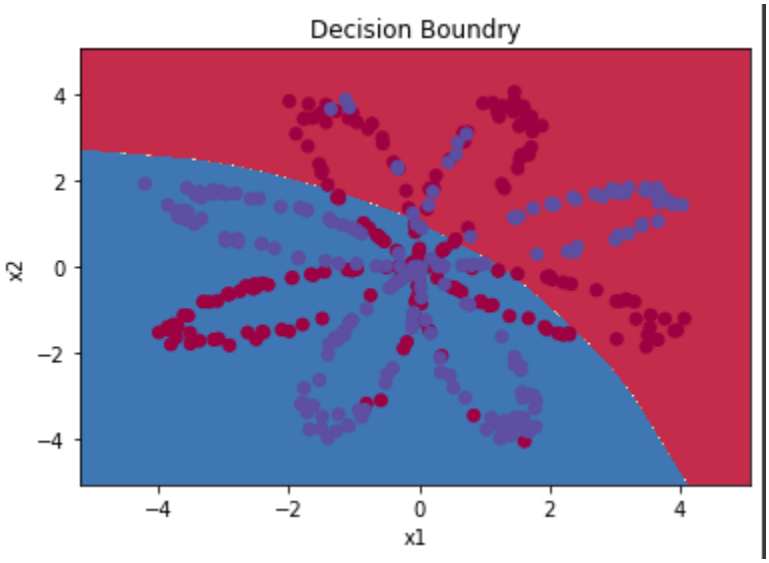
شبکه دچار overfit نشده است چون در واقع نتوانسته با این دو ویژگی به خوبی دیتا را تفکیک کند. وقتی میگوییم overfit رخ داده که دقت روی داده آموزشی به شدت بالا و روی داده تست پایین باشد. در اینجا داده تستی نداریم اما روی داده آموزش هم آنچنان بالا نیست. البته با توجه به روند کاهش loss میتوان فهمید که مشکل بنیادینی وجود ندارد.

تاثیر تعداد لایه‌ها)

hidden_layers=[5,2,2]: (دقت ۵۵.۲۵ درصد)



hidden_layers=[20,5,2,2]: (دقت ۵۹.۲۵ درصد)



اینطور که به نظر میرسد با افزایش تعداد لایه ها به نتایج بهتری میرسیم. چون تفکیک این داده ها با منحنی های پیچیده راحت تر است و با منحنی ساده درجه ۲ یا خطی با دقت کمی قابل تفکیک هستند. البته تعداد نورون های لایه ها را نباید خیلی زیاد کنیم چون لایه ورودی صرفا دو ویژگی دارد. مثلا اگر لایه ها را [100,200,100] میگذاشتیم بیهوده محاسبات را زیاد میکردیم و شاید دچار overfit میشدیم.

مقایسه تاثیر لایه‌ها با نورون‌ها) تعیین تعداد لایه‌ها و نورون ها در شبکه امری بسیار مهم است و شاید بتوان گفت مهمترین هایپرپارامتر شبکه می باشد. این امر به تعداد ویژگی هایی که میخواهیم بررسی کنیم و میزان پیچیدگی پراکندگی دیتاها بازمی گردد. اگر داده ها به صورت خطی قابل تفکیک باشند استفاده از یک لایه هم کافیست. اگر تعداد ویژگی های اولیه مثلا ۲۰۰ باشد، معمولا تعداد نورون های لایه اول مخفی را عددی حدود ۲۰۰ یا کمی کمتر باید بگیریم. و معمولا در هر لایه تعداد نورون ها را باید کاهش دهیم تا در نهایت لایه خروجی به اندازه تعداد کلاس هایی که میخواهیم دیتا را به آن تقسیم کنیم باید نورون داشته باشد.

به این خاطر این پارامتر مهم است که اگر تعداد لایه ها و یا تعداد نورون های لایه بیش از حد زیاد باشد، شبکه دچار overfit میشود. یعنی بسیار حساس به داده آموزشی و با شکل خطی عجیبی آن ها را تفکیک میکند و معمولا به دقت عالی روی داده تمرینی میرسد. اما اگر تعدادشان خیلی نسبت به ویژگی های ورودی و پراکندگی دیتا کم باشد، دچار underfit میشویم. یعنی میخواهیم داده های بسیار پیچیده ای را با یک خط ساده جدا کنیم.

پاسخ سوال پرسیده شده: حالت یک لایه مخفی با ۱۰۰۰ نورون از نظر زمان آموزش طولانی تر از دو لایه با ۲۰۰ نورون است.

همچنین با افزایش تعداد لایه ها انگار آموختن هر ویژگی را به یک لایه میسپاریم و هرچه لایه ها را جلو میرویم ویژگی های جزئی تر آموخته می شود. البته باید ببینیم در این سوال تعداد ویژگی ها و تفکیک داده ها بر اساس آن ها چطور است. ممکن است با لایه های زیاد، شبکه overfit کند.

از نظر حافظه تفاوت چندانی بین دو روش وجود ندارد که قابل بحث باشد اما ذخیره ۴۰۰ وزن طبیعتا حافظه کمتری از ۱۰۰۰ تا نیاز دارد.

۳. پاسخ سوال سوم

پس از پیاده سازی با استفاده از tensorflow.keras، در حالات مختلف دقت شبکه روی داده آموزشی و تست را امتحان کرده و تحلیل میکنیم.

Momentum) دو حالت خواسته شده را با ثابت نگه داشت تمام پارامترهای دیگر شبکه مقایسه میکنیم. در حالتی که صفر باشد:

```
1875/1875 [==============================] - 3s 2ms/step - loss: 0.2313 - accuracy: 0.9183
train accuracy: 91.83499813079834 %
313/313 [==============================] - 0s 1ms/step - loss: 0.3343 - accuracy: 0.8803
test accuracy: 88.0299985408783 %
```

در حالتی که ۰.۹ باشد:

```
1875/1875 [==============================] - 2s 1ms/step - loss: 0.1216 - accuracy: 0.9554
train accuracy: 95.54499983787537 %
313/313 [==============================] - 1s 2ms/step - loss: 0.3994 - accuracy: 0.8869
test accuracy: 88.6900007724762 %
```

اولین چیزی که به چشم می آید سریعتر همگرا شدن شبکه است. میزان کاهش loss و افزایش accuracy در هر epoch، در حالتی که momentum داشته باشیم سریعتر است. میبینیم که دقت نهایی هم بهتر شده است.

چون درواقع دلیل استفاده از momentum این است که نتایج قبلی حاصل شده از گرادیان ها را هم در تا به محاسبات جدید لحاظ کنیم. یعنی جهتی که تا به حال وزن و پایه ها را آپدیت میکردیم را میدانیم و اگر تغییر در آن جهت باشد سرعت تغییر مدام بیشتر میشود. (مانند شیب دار بودن) البته این که مقدار آن را چقدر بگذاریم مانند سایر hyperparameterها مهم است و ۰.۹ عدد مشهور و پرکاربردی برای آن میباشد.

Weight decay) دو حالت خواسته شده را با ثابت نگه داشت تمام پارامترهای دیگر شبکه مقایسه میکنیم. در حالتی که نداشته باشیم:

```
1875/1875 [==============================] - 3s 2ms/step - loss: 0.2313 - accuracy: 0.9183
train accuracy: 91.83499813079834 %
313/313 [==============================] - 0s 1ms/step - loss: 0.3343 - accuracy: 0.8803
test accuracy: 88.0299985408783 %
```

در حالتی که به شبکه پارامتر kernel_regularizer='L1' را میدهیم:

```
1875/1875 [==============================] - 3s 1ms/step - loss: 0.9096 - accuracy: 0.7825
train accuracy: 78.2533347606659 %
313/313 [==============================] - 1s 2ms/step - loss: 0.9289 - accuracy: 0.7730
test accuracy: 77.30000019073486 %
```

در حالتی که به شبکه پارامتر kernel_regularizer='L2' را میدهیم:

```
1875/1875 [==============================] - 3s 1ms/step - loss: 0.4736 - accuracy: 0.8536
train accuracy: 85.36166548728943 %
313/313 [==============================] - 1s 2ms/step - loss: 0.5163 - accuracy: 0.8375
test accuracy: 83.74999761581421 %
```

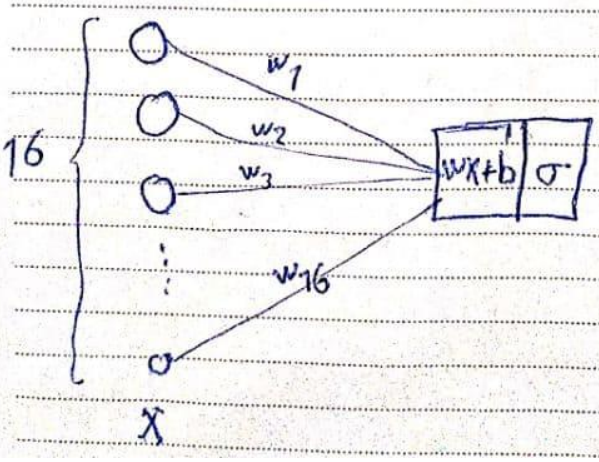
هدف استفاده از weight decay، جلوگیری از بزرگ شدن بیش از اندازه گرادیان ها و در نتیجه آن کوچک شدن وزن هاست. با استفاده از این ویژگی سایز مدل را کاهش میدهیم. هنگامی که شبکه ممکن است overfit کند استفاده از آن توصیه می شود.

در اینجا شبکه ما با تعداد epoch برابر با استفاده از weight decay عملکرد ضعیف تری داشته است. زیرا مقدار ارور با استفاده از آن کم میشود، لذا گرادیان ها کمتر شده و وزن ها کوچک تر می شوند و دیرتر آپدیت میشوند. اما وقتی شبکه ممکن است overfit کند و به خصوص در شبکه های عمیق میتواند مفید باشد.

۴. پاسخ سوال چهارم

دو الگوی بالا را کلاس صفر و دو الگوی پایین را کلاس 1 در نظر می‌گیریم.
چون فقط دو کلاس داریم (مسئله باینری) و تعداد ویژگی‌ها فقط 16 تاست استفاده از یک پرسپترون به نظر کافی می‌آید. چون این مسئله به صورت خطی قابل تفکیک است.
هر کدام از پیکسل‌ها را یک ویژگی در نظر می‌گیریم و بردار ویژگی ما به طول 16 خواهد بود.
از مربع بالا سمت چپ که خانه اول بردار ویژگی‌هاست تا پایین سمت راست را خانه اول در نظر می‌گیریم. پس 4 داده تمرینی داریم با 16 ویژگی که هرکدام وزنی خواهند داشت. و یک bias هم نیاز داریم.
از تابع فعالسازی sigmoid برای binary classification استفاده می‌کنیم. تابع ضرر را MSE، نرخ آموزش را 0.5 (تا شبکه زود همگرا شود) و epochs = 2 در نظر می‌گیریم.
نیازی به batch هم نداریم چون داده زیادی نداریم. ساختار شبکه:

$\eta = 0.5$, epoch = 2



مقادیر اولیه وزن‌ها رندوم و بایاس را صفر در نظر می‌گیریم:

$$w = [0.1, 0.8, 0.7, 0.2, 0.9, 0.3, 0.2, 0.8, 0.9, 0.3, 0.2, 0.8, 0.4, 0.5, 0.6, 0.3]$$

$$b = 0$$

با نرون مربع‌های آبی 1 و مربع‌های سفید صفر:

$$X_1 = [1,0,0,1,0,1,1,0,0,1,1,0,1,0,0,1] \quad , \quad Y_1 = 0$$

داده اول

$$X_2 = [1,0,0,1,0,1,1,0,0,1,1,0,0,1,1,0] \quad , \quad Y_2 = 0$$

$$X_3 = [0,1,1,0,1,0,0,1,1,0,0,1,1,0,0,1] \quad , \quad Y_3 = 1$$

$$X_4 = [0,1,1,0,1,0,0,1,1,0,0,1,0,1,1,0] \quad , \quad Y_4 = 1$$

حالا که همه چیز را داریم شروع به آموزش شبکه می‌کنیم (محاسبات با ماشین حساب):

$$\left.\begin{aligned} Z &= w \cdot X + b \\ A &= \sigma(Z) \end{aligned}\right\} \Rightarrow A_1 = 0.88 \, , \, A_2 = 0.91 \, , \, A_3 = 0.996 \, , \, A_4 = 0.997$$

و حالا نوبت به backpropagation می‌رسد:

$$\frac{dL}{dw} = \frac{dL}{dA} \times \frac{dA}{dZ} \times \frac{dZ}{dw} = (A - Y)(1 - A)A \cdot X$$

مشتق

طبق قاعده زنجیره‌ای

حال باید طبق تابع MSE، مجموع همه گرادیان‌ها را به دست آوریم و برحسب آن‌ها

20 January 2018

که 4 تا است تقسیم کنیم. با انجام این محاسبات dw که برداری هم اندازه w است به دست می آید. db هم عادی محاسبه می شود. سپس آن ها را با این فرمول آپدیت می کنیم:

$$w = w - \eta\, dw \quad , \quad b = b - \eta\, db$$

در نهایت وزن ها و bias به این شکل تغییر می کنند:

$$w = [0.019, 0.81, 0.71, 0.12, 0.91, 0.22, 0.12, 0.81, 0.91, 0.22, 0.12, 0.81, 0.91, 0.22, 0.12, 0.81, 0.35, 0.46, 0.565, 0.253]$$

$$b = -0.082$$

حال برای epoch دوم، همان مراحل قبل را مجدداً از اول تکرار می کنیم. وزن ها و bias به این شکل آپدیت می شوند:

$$b = -0.258$$

$$w = [-0.158, 0.8, 0.7, -0.058, 0.9, 0.04, -0.058, 0.8, 0.9, 0.042, -0.058, 0.8, 0.26, 0.39, 0.479, 0.16]$$

حال داده تست آخر را به شبکه می دهیم:

$$X_{test} = [1,1,1,1,0,0,1,0,0,1,0,0,1,1,1,1] \Rightarrow Z_{test} \approx 2.3$$

$$A_{test} \approx 0.90 \xrightarrow{\text{threshold} = 0.5} \text{داده تست از گروه 1 است.}$$