به نام خدا

گزارش تمرین اول مبانی هوش

غزل زمانی نژاد 97522166

1. در این سوال می خواهیم تابع nor را به کمک یک perceptron طراحی کنیم. بدین منظور، ابتدا یک کلاس SGD طراحی می کنیم که در constructor آن دیتاست، نرخ یادگیری[1] و بیشترین مقدار تغییرات وزن (برای اینکه مشخص شود الگوریتم چقدر به یادگیری ادامه دهد) مقداردهی می شوند. هم چنین یک آرایه برای ذخیره وزن ها به صورت رندوم initialize می شود. (اندازه آرایه از ورودی ها یکی بیشتر است چون ورودی ثابت bias هم درنظر گرفته شده است).

در تابع iterations، فرآیند یادگیری انجام می شود. برای پیاده سازی الگوریتم gradient descent به صورت stochastic، در هر iteration، یک دیتا را بررسی کرده و وزن ها را آپدیت می کنیم. حلقه while تا زمانی ادامه می یابد که بیشترین میزان تغییرات وزن از مقداری که کاربر وارد کرده کمتر شود. در هر iteration ابتدا ایندکسی از دیتاست که باید بررسی شود را می یابیم. سپس مقدار ثابت bias را به ابتدای آن اضافه می کنیم. سپس مقدار خطا را به کمک تابع error می یابیم. مطابق فرمول، مقدار ارور برابر است با:

$$e(n) = d(n) - \sum_{j=0}^{m} x_j(n) w_j(n)$$

در این فرمول مقدار d را به کمک تابع desired محاسبه می کنیم. برای اینکه عدد 0 در محاسبات (خصوصا ضرب) خللی ایجاد نکند، به جای 0، از 1- استفاده می کنیم. پس خروجی تابع nor تنها در حالتی که همه ورودی ها 1- باشند، 1 می شود. بعد از محاسبه d، حاصل ضرب داخلی دو بردار را می یابیم و این دو مقدار را از هم کم می کنیم.

اکنون باید آرایه وزن را طبق فرمول زیر آپدیت کنیم:

$$w(n+1) = w(n) + \eta x(n) e(n)$$

[1] Learning rate

بعد از محاسبه خطا، آن را در learning rate ای که کاربر وارد کرده بود ضرب می کنیم. و مقدار حاصل را در همه ی المان های آن دیتا ضرب می کنیم. در پایان مقدار add_term را به آرایه وزن اضافه می کنیم. بیشترین میزان تغییرات را ذخیره می کنیم تا بدرستی تشخیص دهیم الگوریتم تا چه زمانی ادامه یابد. در پایان هر iteration، مقدار learning rate را در عددی ضرب می کنیم تا کاهش یابد (در غیر این صورت ممکن است وارد حلقه بی نهایت شویم).

سپس با یک مثال الگوریتم را امتحان می کنیم:

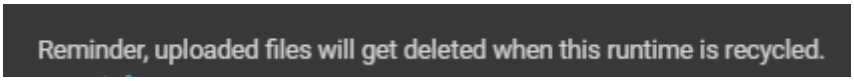
```
data = np.array([[-1, -1],
                 [-1, 1],
                 [1, -1],
                 [1, 1]])

gd = StochasticGradientDescent(data, 0.3, 0.0001) #(n, eta, small number)
i = gd.iterations()
print("Learning is done in ", i, " itetation.")
print("Weights are: ", gd.weights)
```

```
Learning is done in  7311  itetation.
Weights are:  [-0.50009994 -0.50004997 -0.50000005]
```

در این مثال، تعداد نورون های ورودی 2، اندازه دیتاست 4، learning rate 0.3 و بیشترین میزان تغییرات 0.001 در نظر گرفته شده است. این مقادیر در آزمون و خطا بدست آمده اند.

2. در این سوال میخواهیم مسئله Binary Classification را پیاده سازی کنیم. ابتدا فایل دیتا را در محیط colab آپلود می کنیم. (فایل های آپلود شده بعد از recycle شدن runtime از بین می روند به همین دلیل باید دوباره آپلود کنیم.)



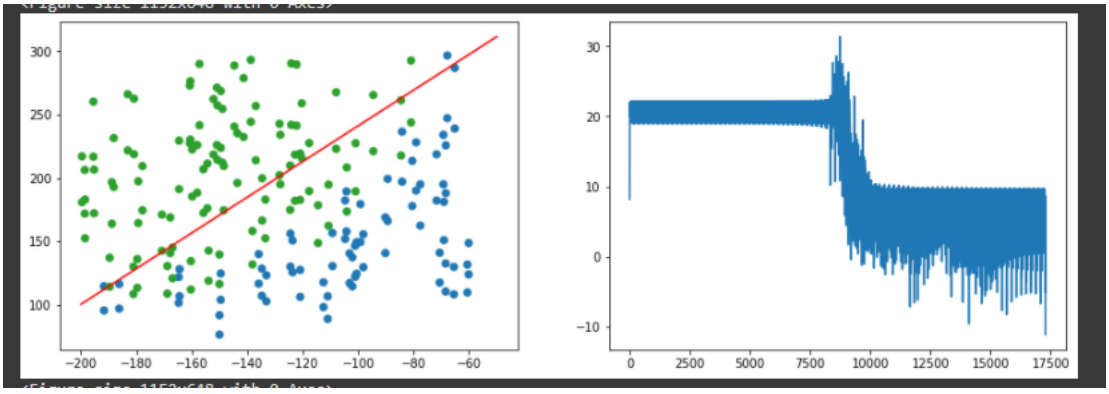
بعد فایل را با دستور open باز کرده و با دستور readline خطوط آن را می خوانیم. تک به تک خطوط را اسپلیت کرده و در آرایه های دیتا، دیتا همراه با bias و desired ذخیره می کنیم. سپس با استفاده از کلاس stochasticGradientDescent به حل این سوال می پردازیم. در constructor این کلاس دیتا با ثابت bias، desired، نرخ یادگیری و بیشترین میزان تغییرات وزن را ذخیره می کنیم. و مقادیر وزن را به صورت رندوم مقداردهی می کنیم. بعد با تابع iterations در حلقه به آپدیت کردن وزن می پردازیم. تمامی مراحل مشابه سوال قبلی است. با این تفاوت که در هر مرحله، باید وزن ها را به مقدار مشخصی تقسیم کنیم تا مقادیر وزن بزرگ (inf, nan) نشوند و هم چنان قابل محاسبه باشند. در هر لوپ، مقادیر ارور را مطابق فرمول زیر محاسبه و ذخیره می کنیم:

$$E(w(n)) = \frac{1}{2}e^2(n)$$

حلقه تابع iterations، زمانی متوقف می شود که مقدار تغییر وزن از مقداری که کاربر وارد کرده کمتر شود. در پایان به رسم نمودارها می پردازیم. مطابق فرمول زیر، شیب و عرض از مبدا خط را محاسبه می کنیم:

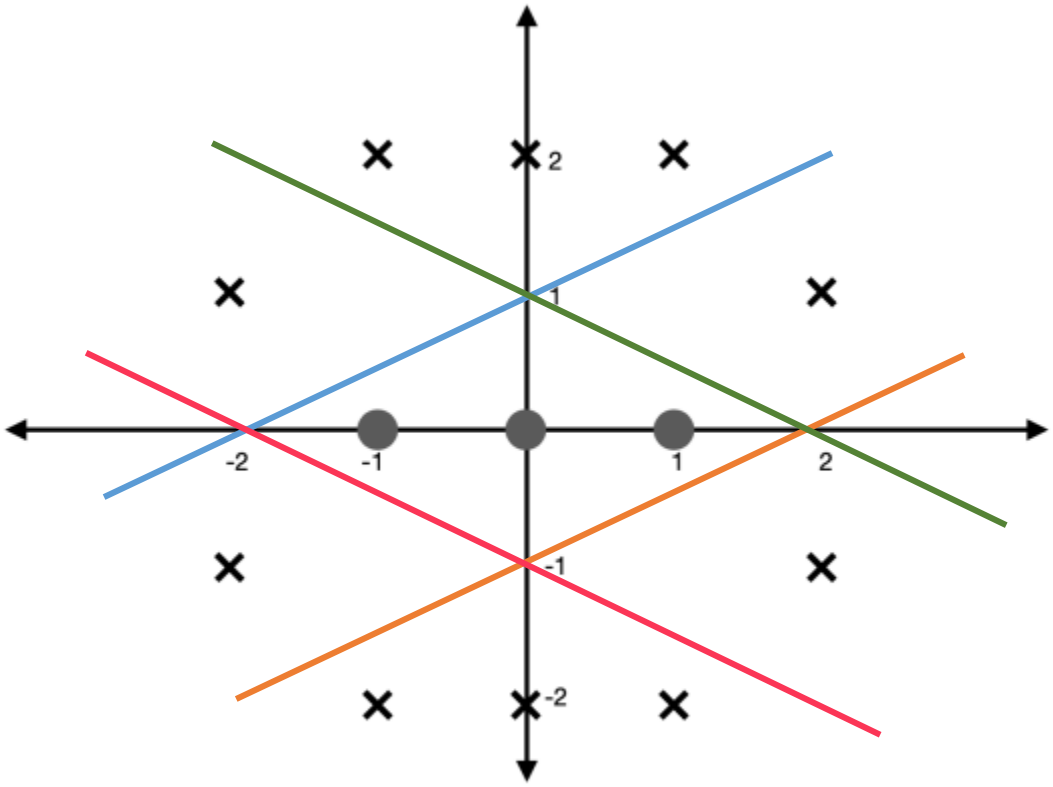
$$w_1x_1 + w_2x_2 + b = 0$$

سپس با استفاده از linspace ابتدا و انتهای بازه روی محور افقی و تعداد نقاط مد نظر را مشخص می کنیم. به نقاط ورودی مطابق دسته بندی کلاس آن با آرایه colors رنگ می دهیم. Figure را به دو subplot تقسیم بندی می کنیم. با تابع scatter نقاط ورودی را رسم می کنیم. و با تابع plot خطی که نقاط را دسته بندی می کند و همچنین تابع خطا را رسم می کنیم:



3. زمانی که چند Adaline را به طور موازی پیاده سازی و استفاده کنیم، به Madaline می رسیم. Madaline می تواند مسائلی را حل کند که تفکیک پذیر خطی نیستند ولی دسته بندی آن را می توان به کمک ناحیه ای محدب انجام داد. نحوه کار آن به این صورت است که جواب چند Adaline را and می کند و با توجه به نتیجه and (که نتیجه همه ی Adalineها در آن دخیل است) خروجی می دهد. در پایان اگر دسته بندی به کمک یک ناحیه محدب قابل انجام باشد، خروجی Madaline جواب مدنظر را می دهد. داخل محدوده، یک کلاس دسته بندی و خارج از آن کلاس دیگر است.

آ) این مسئله به کمک Madaline قابل حل است چون به کمک خطوط زیر (که از چند Adaline بدست آمده اند) به ناحیه ای محدب دست پیدا می کنیم.



(آ)

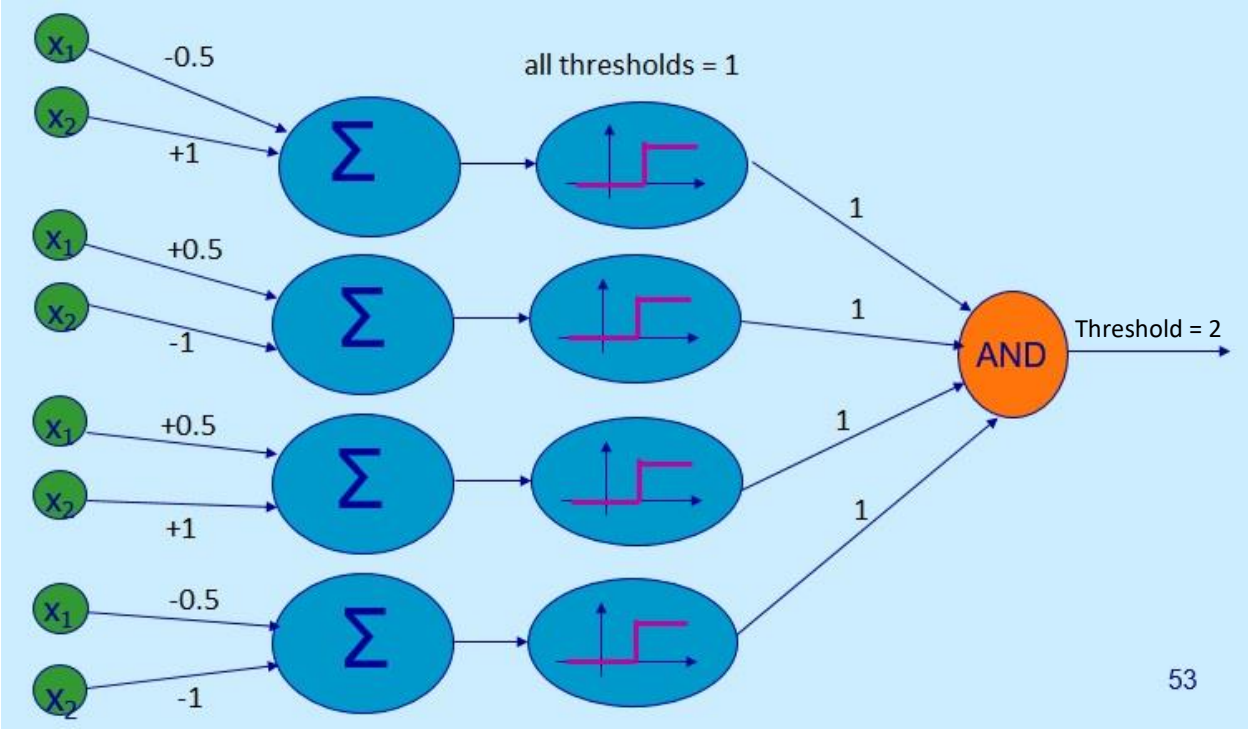
این شبکه عصبی، ناحیه ای محدب تشکیل داده که توانسته به خوبی نقاط دایره را از مربع تفکیک کند.

به بررسی ساختار این شبکه عصبی می پردازیم:

معادله خطوطی که از Adalineها بدست آمده اند:

$$w_1x_1 + w_2x_2 + b = 0$$

- Blue: y = 0.5 x + 1 → w1 = -0.5 , w2 = +1, b = -1, threshold = +1;
- Orange: y = 0.5 x – 1 → w1 = +0.5 , w2 = -1, b = -1, threshold = +1;
- Green: y = -0.5 x + 1 → w1 = +0.5 , w2 = +1, b = -1, threshold = +1;
- Pink: y = -0.5 x – 1 → w1 = -0.5 , w2 = -1, b = -1, threshold = +1;



ب) این مثال به کمک تنها یک Madaline قابل حل نیست. برای جداسازی نقاط دایره شکل از نقاط ضربدر به دو Madaline نیاز داریم که دو ناحیه محدب داشته باشیم.

4. در این سوال به کمک کتابخانه keras، به دسته بندی داده های mnist می پردازیم. ابتدا با استفاده از mnist.load_data، داده های learn و تست را وارد می کنیم. داده های ایمپورت شده 2بعدی هستند، ابتدا آن ها را در یک آرایه یک بعدی ذخیره می کنیم. داده ها را با فرمول زیر نرمالایز می کنیم.

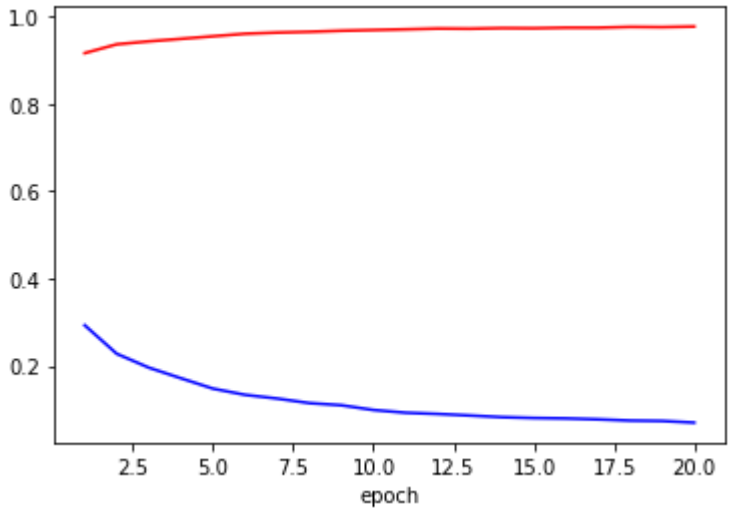
X => (X - Xmin)/(Xmax-Xmin) = X/255

در اینجا چون x_{min} 0 است، کافیست داده ها را به 255 تقسیم کنیم.

بعد outputها را به صورت one hot، encode می کنیم. یعنی یک آرایه با اندازه 10 در نظر گرفته، ایندکسی که نشان دهنده y است را 1 و سایر ایندکس ها را 0 مقداردهی می کنیم.

اکنون با استفاده از تابع sequential (مربوط به keras)، یک MLP تشکیل می دهیم. با تابع Dense، لایه با مشخصات مورد نظر را می سازیم و به پرسپترون چندلایه add می کنیم. لایه ها با آزمون و خطا به صورت زیر بدست آمده اند:
لایه اول دارای 512 نورون و لایه دوم دارای 128 نورون است و activation function هر دو لایه ReLu می باشد.
لایه خروجی نیز دارای 10 نورون است و از softmax به عنوان activation function استفاده می کند.
بعد از تشکیل MLP، آن را compile می کنیم. در این مثال از SGD به عنوان optimizer استفاده شده، از categorical_crossentropy برای خطا استفاده شده است.
اکنون MLP را با استفاده از متد fit، train می کنیم. ورودی های این تابع به صورت زیر هستند:
دیتای ورودی، دسته بندی دیتای ورودی، تعداد epoch (تعداد iteration بر روی دیتای ورودی) و در نهایت دیتایی که می خواهیم روی آن دقت و خطا را محاسبه کنیم (در واقع همان دیتای تست).
در آخر accuracy و lossای که از دیتای تست بدست آمده را رسم می کنیم.

```
Epoch 1/20
1875/1875 [==============================] - 9s 5ms/step - loss: 0.9244 - accuracy: 0.7644 - val_loss: 0.2930 - val_accuracy: 0.9166
Epoch 2/20
1875/1875 [==============================] - 8s 5ms/step - loss: 0.2851 - accuracy: 0.9196 - val_loss: 0.2276 - val_accuracy: 0.9368
Epoch 3/20
1875/1875 [==============================] - 9s 5ms/step - loss: 0.2229 - accuracy: 0.9368 - val_loss: 0.1958 - val_accuracy: 0.9436
Epoch 4/20
1875/1875 [==============================] - 9s 5ms/step - loss: 0.1859 - accuracy: 0.9466 - val_loss: 0.1715 - val_accuracy: 0.9493
Epoch 5/20
1875/1875 [==============================] - 9s 5ms/step - loss: 0.1596 - accuracy: 0.9539 - val_loss: 0.1477 - val_accuracy: 0.9552
Epoch 6/20
1875/1875 [==============================] - 9s 5ms/step - loss: 0.1401 - accuracy: 0.9594 - val_loss: 0.1335 - val_accuracy: 0.9610
Epoch 7/20
1875/1875 [==============================] - 9s 5ms/step - loss: 0.1241 - accuracy: 0.9652 - val_loss: 0.1250 - val_accuracy: 0.9637
Epoch 8/20
1875/1875 [==============================] - 9s 5ms/step - loss: 0.1073 - accuracy: 0.9697 - val_loss: 0.1146 - val_accuracy: 0.9654
Epoch 9/20
1875/1875 [==============================] - 9s 5ms/step - loss: 0.0991 - accuracy: 0.9725 - val_loss: 0.1097 - val_accuracy: 0.9679
Epoch 10/20
1875/1875 [==============================] - 9s 5ms/step - loss: 0.0891 - accuracy: 0.9757 - val_loss: 0.0988 - val_accuracy: 0.9695
Epoch 11/20
1875/1875 [==============================] - 9s 5ms/step - loss: 0.0820 - accuracy: 0.9780 - val_loss: 0.0925 - val_accuracy: 0.9711
Epoch 12/20
1875/1875 [==============================] - 9s 5ms/step - loss: 0.0746 - accuracy: 0.9798 - val_loss: 0.0897 - val_accuracy: 0.9729
Epoch 13/20
1875/1875 [==============================] - 9s 5ms/step - loss: 0.0663 - accuracy: 0.9821 - val_loss: 0.0862 - val_accuracy: 0.9725
Epoch 14/20
1875/1875 [==============================] - 9s 5ms/step - loss: 0.0626 - accuracy: 0.9839 - val_loss: 0.0824 - val_accuracy: 0.9739
Epoch 15/20
1875/1875 [==============================] - 9s 5ms/step - loss: 0.0595 - accuracy: 0.9837 - val_loss: 0.0804 - val_accuracy: 0.9736
Epoch 16/20
1875/1875 [==============================] - 9s 5ms/step - loss: 0.0533 - accuracy: 0.9854 - val_loss: 0.0794 - val_accuracy: 0.9748
Epoch 17/20
1875/1875 [==============================] - 9s 5ms/step - loss: 0.0500 - accuracy: 0.9870 - val_loss: 0.0773 - val_accuracy: 0.9748
Epoch 18/20
1875/1875 [==============================] - 9s 5ms/step - loss: 0.0476 - accuracy: 0.9873 - val_loss: 0.0745 - val_accuracy: 0.9767
Epoch 19/20
1875/1875 [==============================] - 9s 5ms/step - loss: 0.0437 - accuracy: 0.9890 - val_loss: 0.0736 - val_accuracy: 0.9762
Epoch 20/20
1875/1875 [==============================] - 9s 5ms/step - loss: 0.0416 - accuracy: 0.9889 - val_loss: 0.0698 - val_accuracy: 0.9775
```



نمودار آبی نشان دهنده خطا در هر epoch است که به مرور کاهش یافته و نمودار قرمز نشان دهنده میزان دقت در هر epoch است که بعد از 20 epoch به 1 بسیار نزدیک شده است.

5. در این سوال یک MLP با 3لایه طراحی کرده ایم. مراحل اصلی کار به صورت زیر است:
ساختن شبکه عصبی، forward pass، backward pass، آموزش روی دیتای ورودی، تست
به توضیح هریک از مراحل می پردازیم:

- ✓ ساختن شبکه عصبی: در constructor کلاس MLP، تعداد نورون های لایه ورودی، تعداد نورون های لایه میانی، تعداد نورون های لایه خروجی، دیتای ورودی و نرخ یادگیری مقداردهی می شوند. وزن های میان لایه ورودی_پنهان و لایه پنهان_خروجی و هم چنین ثابت های bias به صورت رندوم مقداردهی می شوند. ورودی، پیکسل های 28*28 است، پس به 784 نورون ورودی نیاز داریم. خروجی یک رقم است، آن را به صورت one hot کد میکنیم؛ یعنی یک آرایه پرشده با 0 به طول 10 در نظر می گیریم و ایندکسی از آن که برابر با خروجی است را به 1 تغییر می دهیم. برای لایه میانی 128 نورون در نظر می گیریم.
پس دو آرایه وزن به صورت numpy array و با سایزهای (784 ,128) و (128 ,10) درنظر می گیریم.
- ✓ forward pass: ابتدا حاصل ضرب داخلی بردار w و data را محاسبه کرده، با bias جمع می کنیم. بعد بردار بدست آمده را به عنوان ورودی به activation function می دهیم.
activation function لایه ورودی_پنهان را sigmoid، لایه پنهان_خروجی را softmax درنظر گرفته ایم.
تابع sigmoid:

$$\varphi(\mathrm{v_j}) = \frac{1}{1+e^{-av_j}} \quad \text{with } a > 0$$

خروجی که از تابع sigmoid بدست می آید را در out1 ذخیره می کنیم و به عنوان ورودی لایه بعدی در نظر می گیریم. حاصل ضرب داخلی بردار wPrime و out1 را محاسبه می کنیم، با bias جمع می کنیم. بردار بدست آمده را به softmax می دهیم.

تابع softamx:

$$S(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{N} e^{z_j}}$$

در پایان هر 4آرایه (حاصل سیگمای لایه اول، خروجی لایه اول، حاصل سیگمای لایه دوم، خروجی لایه دوم) را ریترن می کنیم.

- ✓ backward pass: در این مرحله باید ارور را back propagate کنیم. برای این کار ابتدا به محاسبات ریاضی آن می پردازیم.
  برای آنکه در هر مرحله شبکه عصبی دقیق تر عمل کند، باید وزن ها را آپدیت کنیم. مقدار تغییر وزن را به گونه ای محاسبه می کنیم که در هر مرحله مقدار ارور کاهش یابد.

- ✓ مطابق gradient descent:

$$w_{ji} = w_{ji} + \Delta w_{ji}$$

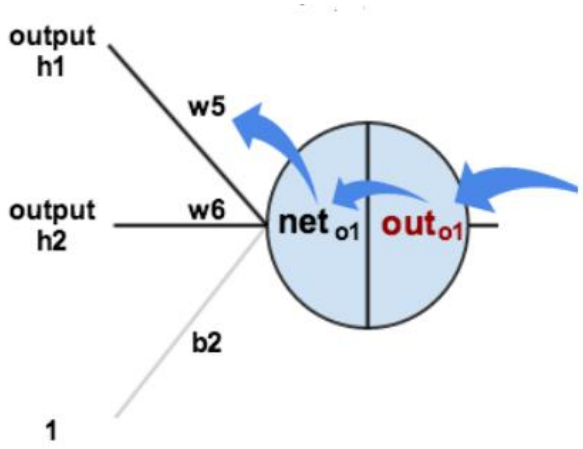
$$\Delta w_{ji} = -\eta \frac{\partial E}{\partial w_{ji}}$$

اکنون مشتقات جزئی E نسبت به هر یک از وزن ها را باید حساب کنیم. نحوه محاسبه برای لایه پنهان_خروجی با لایه ورودی_پنهان متفاوت است.
نحوه انجام این محاسبات مطابق زیر است:

(حاصل سیگما در هر لایه = net)

(حاصل activation function در هر لایه = out)

- لایه خروجی


output
h1
w5
output
h2
w6
net o1
out o1
b2
1

$$\frac{\partial E_{total}}{\partial w_i} = \underbrace{\frac{\partial E_{total}}{\partial out_{o_j}}}_{(I)} \times \underbrace{\frac{\partial out_{o_j}}{\partial net_{o_j}}}_{(II)} \times \underbrace{\frac{\partial net_{o_j}}{\partial w_i}}_{(III)}$$

$$(I)\quad \frac{\partial E_{total}}{\partial out_{o_j}} = \frac{\partial\left(\frac{1}{2}\sum_i (d_i - out_i)^2\right)}{\partial out_{o_j}} = (d - out_j)(-1) = out_j - d$$

$$(II)\quad \sigma(o_j) = \frac{e^{o_j}}{\sum_{i=1}^{N} e^{o_i}} \quad \frac{\partial}{\partial(o_j)} \quad \frac{e^{o_j}\left(\sum_{i=1}^{N} e^{o_i}\right) - e^{o_j} \times e^{o_j}}{\left(\sum_{i=1}^{N} e^{o_i}\right)^2} =$$

$$= \frac{e^{o_j}\left(\sum e^{o_i} - e^{o_j}\right)}{\left(\sum_{i=1}^{N} e^{o_i}\right)^2} = \frac{e^{o_j}}{\sum e^{o_i}} \times \left(1 - \frac{e^{o_j}}{\sum e^{o_t}}\right)$$

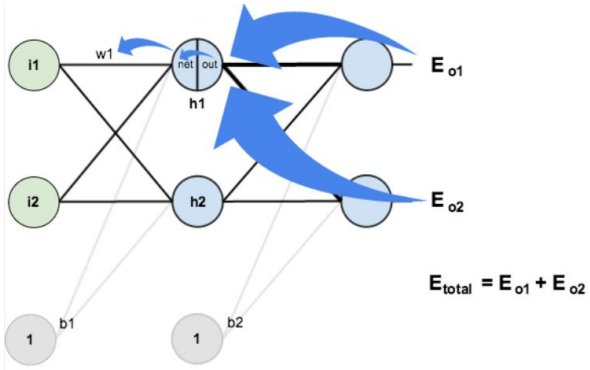
$$= \sigma(o_j)\left(1 - \sigma(o_j)\right)$$

$$(III)\quad \frac{\partial net_{o_j}}{\partial w_i} = \frac{\partial\left(\sum_k w_k L_k + b\right)}{\partial w_i} = w_i$$

$$\Rightarrow \frac{\partial E_{total}}{\partial w_i} = \left(out_j - d\right)\sigma(o_j)\left(1 - \sigma(o_j)\right) w_i$$

لایہ خروجی

لایه میانی:



Sunday یکشنبه
۲۰ دیقعده
FEB

$$\frac{\partial E_T}{\partial w_i} = \frac{\partial E_T}{\partial out_{h_j}} \times \frac{\partial out_{h_j}}{\partial net_{h_j}} \times \frac{\partial net_{h_j}}{\partial w_i}$$

(1) $$\frac{\partial E_T}{\partial out_{h_j}} = \sum_o \frac{\partial E_T}{\partial out_o} \times \frac{\partial out_o}{\partial net_o} \times \frac{\partial net_o}{\partial out_{h_j}} =$$

$$= \sum_o (I) \times (II) \times \frac{\partial (\sum_k w_k h + b)}{\partial out_{h_j}}$$

$$= \sum_o (I) \times (II) \times w_o$$

(2) $$G(h_j) = \frac{1}{1+e^{-net_{h_j}}} \quad \frac{\partial}{\partial net_{h_j}} \; G(h_j)(1 - G(h_j))$$

(3) $$\frac{\partial net_{h_j}}{\partial w_i} = \frac{\partial}{\partial w_i} \left(\sum_k w_k i + b \right) = i_{w_i}$$

$$\frac{\partial E_T}{\partial w_i} = \left[\sum_o (I)(II) w_o \right] \times G(h_j)(1 - G(h_j)) \times i_{w_i}$$

لایه میانی

تمامی این محاسبات در تابع deltachange با استفاده از numpy انجام شده است. وزن های بایاس ها را نیز به همین شکل آپدیت میکنیم.

- ✓ آموزش روی دیتای ورودی: با تابع train روی هر epoch از دیتاست عملیات یادگیری را انجام می دهیم. هر خط از ورودی را اسپلیت می کنیم. برای هر خط از ورودی، تابع forwardbackward را صدا می زنیم که در آن ابتدا دیتا نرمالایز شده، سپس عملیات فوروارد بر روی آن انجام شده و در نهایت با انجام عملیات backward وزن ها آپدیت شده اند.
  این کار را برای تعداد مشخصی epoch انجام می دهیم.

- ✓ تست: در این مرحله ابتدا دیتای تست خوانده شده. آن را به تابع test پاس می دهیم. ابتدا اسپلیت می کنیم. سپس یک بار عملیات فوروارد را انجام می دهیم تا مقدار نورون های خروجی را بیابیم. بعد از یافتن جواب، آن را با مقدار مورد انتظار مقایسه می کنیم. برای بررسی میزان درستی از binary accuracyدر صورتی برابری، یکی به مقدار true اضافه می کنیم.

پس از یک دور ران کردن نتایج را مشاهده می کنید: