شبکه عصبی شامل دو بخشه:

Feedforward:

1. شامل ضرب ورودی ها در وزن : wx= w*x
2. پیش بینی خروجی با دادن مقدار WX به تابع فعالساز که در اینجا SIGMOID هست: y_pred = sigmoid(wx)

Back prppagation:

1. محاسبه تابع هزینه که در اینجا تابعش mse یا همون 0.5 * (y_pred – label) است
2. کمینه کردن تابع هزینه که برای کمینه کردنش باید وزن ها و بایاسی پیدا کنیم که مقدار کمیته رو بهمون بدن که ما از الگوریتم گرادیان کاهشی استفاده میکنیم به این صورت که هر بار وزن ها رو به شکل زیر اپدیت میکنیم:

باید مشتق تابع هزینه رو نسبت به وزن ها بدست بیاریم

W = w – lr * d_cost/d_dw

- گرادیان کاهشی در واقع همان مشق گرفتن ازمعادله است ما هنگامی که میخواهیم کمینه یک تابع را بدست اوریم از ان تابع مشتق گرفته و برابر با صفر قرار میدهیم . به همین دلیل از هزینه مشتق میگیریم و در خلاف جهت آن میزان wرا تغییر میدهیم.

تمرین اول: داخل فایل tamrin1.py داده ها با استفاده از کتابخانه sklearn به دو دسته آموزشی و تست تقسیم شده اند و با کتابخانه matplotlib به نمایش درآمده اند. عکس آن نیز با اسم tamrin1_1.png در فایل موجود است

تمرین دوم:

K = W*X+B

S(x) = 1/(1+exp(-x)), S’(x) = S(x) * (1-S(x))

Y = S(K)

Dcost/dw = dcost/dy * dy/dw

Dcost/dy = (y’-y)

Dy/dw = dS(k)/dk * dk/dw

DS(k)/dk = S(k)*(1-S(k))

Dk/dw = x

Dcost/dw = (y’-y)*S(k)*(1-S(k))*x

کافیه تنها x یا y یک سمت معادله نگه داری تا مقدارشون بدست بیاد.

تمرین سوم و چهارم : در فایل tamrin1.py شبکه مورد نظر پیاده سازی شده است و داده های هایی که به عنوان داده اموزشی و تست در تصویر tamin1_1.png موجود بود به شبکه داده شده است. خروجی آن که شامل کلاس بندی و تابع هزینه می باشد در تصاویر tamrin2_1.png, cost_1.png نمایش داده شده است.

دستور اجرای برنامه:python tarmin1.py

تمرین **پنجم:**

محاسبه پارامتر ها برای شبکه ی دو لایه:

WX = W.X + B1

VX = V.X + B2

Z1 = S(WX), Z2 = S(VX)

Z = [Z1, Z2]

UZ = U.Z + B3

Y_pred = S(UZ)

1. Dcost/du = dcost/dpred * dpred/duz * duz/dz
    1. Dcost/dpred = y_pred – y
    2. Dpred/duz = S'(UZ)
    3. Duz/dz = Z
2. Dcost/db3 = dcost/dpred * dpred/duz
3. Dcost/dw = dcost/dpred * dpred/dzu * dzu/dz1 * dz1/dwx * dwx/dw
    1. Dcost/dpred = y_pred – y
    2. Dpred/dzu = S’(ZU)
    3. Dzu/dz1 = U[0]
    4. Dz1/dwx = S’(Z1)
    5. Dwx/dw = X
4. Dcost/db1 = dcost/dpred * dpred/dzu * dzu/dz1 * dz1/dwx
5. Dcost/dv = dcost/dpred * dpred/dzv * dzu/dz2 * dz2/dvx * dvx/dv
    1. Dcost/dpred = y_pred – y
    2. Dpred/dzu = S’(ZU)
    3. Dzu/dz2 = U[1]
    4. Dz2/dvx = S’(Z2)
    5. Dvx/dv = X
6. Dcost/db2 = dcost/dpred * dpred/dzv * dzu/dz2 * dz2/dvx

جداسازی داده ها و شبکه مورد نظر در tamrin2.py پیاده سازی شده است و میتوان مانند شبکه اول عکس های خروجی را در فایل های tamin1_2.png, tamrin_2_2.png, cost_2.png مشاهده نمود

تحلیل کد:

چرا دقت مناسبی در شبکه اول نداریم؟

هر لایه در شبکه عصبی نشون دهنده یک تابع است در اینجا میبنیم که معادله نود بصورت Y= wx+b هست که اگر دقت کنید این همون معادله خط هست y=ax+b هر خط میتونه فضای دو بعدی مارو به دو قسمت تبدیل کند. اما اگر به نوع پراکندگی داده ها دقت کنید میبینید که با یک خط قایل جدا سازی نیستند و اون خط رو هر جای صفحه هم قرار بدید نمیتواند داده ها رو از هم بدرستی جدا کند. برای جداسازی این داده ها حداقل به دوتا لایه احتیاج داریم که بتونیم دوتا معادله خط داشته باشیم و با ان داده ها را جدا کرد. در قسمت دوم که یک لایه مخفی به شبکه اضافه کرده ایم دقت افزایش و هزینه کاهش میابد و قدرت شبکه در کلاس بندی افزایش یافته است.

توضیح کد:

1. ماتریس وزن W رو اول بصورت رندم مقداردهی میکنیم.
2. یک حلقه به تعداد epoch ها هست برای اینکه چندین بار داده ها رو به شبکه بدیم تا شبکه بخوبی آموزش ببینه(در واقع مقدار W محاسبه بشود)
3. یک حلقه بر روی داده ها و برچسبش ایجاد شده است با توجه به فرمول کلاسی(برچسب) که ما برای اون داده پیش بینی میکنیم برابره میشود با:

   Y_pred = sigmoid(W*X+b)

   که X همون داده ی هست که ما خوندیم و w ماتریس وزن ها. تا اینجا feedforward شبکه بود.
4. حالا خطا رو اندازه گیری میکنیم که برابر هست با کلاسی که ما پیش بینی کرده ایم منهای کلاس واقعی:

Error = y_pred – y

   که y در کد همون برچسب درست هست که از فایل خوندیم در کد به اسم label هست.

   مقدار هزینه هم برابر است با:

0.5 * (y_pred – y)^2

5. حالا باید از هزیته نسبت به وزن ها مشتق بگیریم چون ما میخواهیم وزنی رو بدست بیاریم که هزینه کمینه بشه پس مقدار d_cost/d_w رو محاسبه میکنیم.

6. برای اینکه بتوانیم مشتق بالا رو حساب کنیم از قاعده ی زنجیره ای استفاده میکنیم چون مستقیما مقدارش قابل محاسبه نیست.

d_cost/d_w = d_cost/d_pred * d_pred/d_z * d_z/d_w

1. مقدار d_cost/d_pred برابر میشود با مشتق تابع هزینه که:

   D_cost/d_pred = 0.5 * 2 * (y_pred – y) = (y_pred – y) = error

2. مقدار d_pred/d_z که برابر هست با مشتق تابع sigmoid: (مقدار z رو در شماره ۳ گفتم اگر بخواهیم از مقدار پیش بینی شده نسبت به z مشتق بگیریم. در واقع باید باید از sigmoid مشتق بگیریم چون این تابع هست که مقدار پیش بینی شده(y_pred) رو بر حسب z به ما میده.

   S(x)*(1-S(x))

3. مقدار z برابر هست با wx+b میشه اگر بخواهیم نسبت به w مشتق بگیریم خروجیش میشه x.
4. حاصلضرب مقادیر بالا هم میشه:

   D_cost/d_w = error * S(WX)*(1-S(WX)) * X

7. حالا با توجه به فرمول مقدار w رو اپدیت میکنیم که برابر است با:

W = w-lr*d_cost/dw

- اینکه ارورها رو با هم جمع میکنیم و در انتهای epoch وزن ها رو اپدیت میکنیم به این دلیل هست که اگر بخواهیم با هر محاسبه خطا وزن رو آپدیت کنیم هزینه اجرایی خیلی بالا میره به همین دلیل در انتها اینکار رو انجام میدن. ولی درستش همین هست که بعد از محاسبه خطا backpropagation انجام شود.
- دلیل استفاده از bias: همونطور که گفته شده معادله شبکه معادله یک خط هست y = ax+b اگر پراکندگی داده ها بصورتی باشه که بتوان خطی رو پیدا کرد که از مرکز بگذره و داده ها رو بدرستی جدا کند نیاز به بایاس نیست اما خیلی از مواقع خط از مرکز عبور نمیکند به همین دلیل از بایاس استفاده میشود.

اول و انتهای کد هم از کتابخونه matplotlib برای نمایش داده ها استفاده شده