

تمرین سری سوم **درس مبانی هوش محاسباتی**

**دکتر نوشین مقصودی**

دانشجو : سید علی شهداالحسینی شماره دانشجویی : 97122920012

سوال اول – الگوریتم Backpropagation

**در این سوال قرار است تا به کمک مجموعه داده های Iris یک شبکه عصبی شامل 4 نرون ورودی، 4 نرون در یک لایه پنهان و 3 نرون در لایه خروجی را آموزش دهیم. پیاده سازی این شبکه عصبی به کمک الگوریتم BP می باشد.**

**مقدمه**

در این مسئله قرار است تا ما کمک یادگیری نظارت شده الگوریتم خود را آموزش دهیم. می دانیم که در یادگیری نظارت شده در مرحله آموزش دادن الگوریتم، ما در کنار هر ورودی ای که به الگوریتم می دهیم، خروجی مطلوب و مد نظرمان برای آن ورودی را نیز می دانیم و سعی می کنیم پاسخ الگوریتم را به پاسخ مطلوب و درست مد نظرمان سوق دهیم که اینکار در الگوریتم های شبکه های عصبی به کمک بروزرسانی وزن ها صورت می گیرد.

در حالت کلی در سری الگوریتم های شبکه عصبی تمام چیز هایی که باید آموزش ببینیم وزن نرون ها می باشد؛ زیرا که این وزن ها هستند که ما را برای رسیدن به خروجی و هدف مورد نظرمان کمک می کنند و ما با دانستن خروجی های هر سری ورودی می توانیم اختلاف خطا خروجی مطلوب و خروجی بدست آمده از شبکه را بدست آوریم و نهایتا در جهتی حرکت کنیم که این اختلاف به کمترین حد خود برسد.

برای حل یک مسئله به روش یادگیری نظارت شده ما باید یکسری داده داشته باشیم به عنوان و ورودی و همچنین باید خروجی مطلوب داده ورودی نیز برای ما مشخص باشد تا بتوانیم با دادن ورودی و گذراندن الگوریتم و بدست آوردن خروجی برای هر داده، خروجی به دست آمده شده را با خروجی های از پیش تعریف و مشخص شده آن ورودی مقایسه کنیم و نهایتا الگوریتم و تغییر وزن های شبکه را در جهتی پیش ببریم که این دو خروجی ما تا حد امکان به هم نزدیک شوند و اختلافشان به کمترین میزان ممکن برسد.

**بررسی جزئیات مسئله**

برای حل هر مسئله ای باید ابتدا مسئله را به طور کامل واضح و شفاف برای خودمان مشخص کنیم. در این قسمت به انجام این کار می پردازیم.

سوال را بررسی می کنیم و نکات اصلی آن را لیست می کنیم:

1. پیاده سازی یک شبکه عصبی به کمک الگوریتم Backpropagation

از این قسمت اطلاعات زیر بدست می آید:

* 1. قرار است یکبار کل شبکه طی شود و خروجی بدست آید.
  2. سپس بعد از بدست آوردن خروجی باید به عقب برگردیم و به کمک مشتق توابع فعال ساز، وزن ها را بروزرسانی کنیم.

1. یک شبکه عصبی دو لایه داریم.

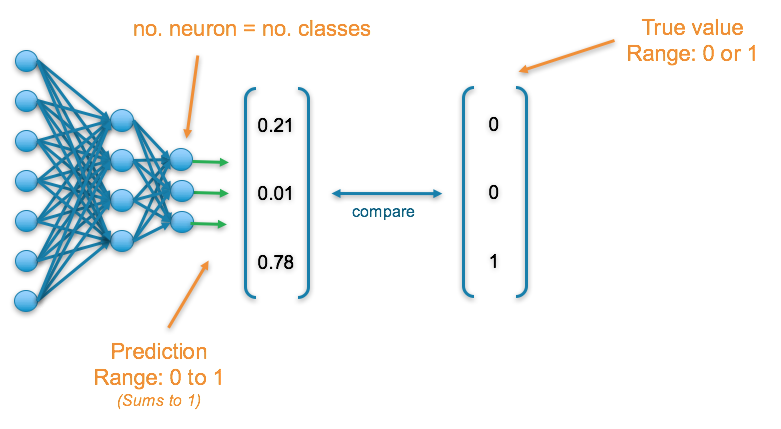
* نکته: در شمارش تعداد لایه های شبکه عصبی، لایه ورودی را در شمارش حساب نمی کنیم، به عبارت دیگر تعداد لایه های شبکه عصبی برابر است با اعداد لایه هایی که مابین لایه ورودی و لایه خروجی هست بعلاوه یک لایه خروجی.

1. در لایه مخفی آن 4 نرون داریم و در لایه خروجی آن 3 نرون داریم.
2. 4 ورودی داریم که در لایه ای قبل از لایه مخفی به عنوان لایه ورودی قرار می گیرد و وارد الگوریتم می شود.
3. تابع فعال ساز لایه مخفی، تابع Sigmoid می باشد.
4. از این قسمت با توجه BP بودن شبکه ما، به این نتیجه می رسیم که باید مشتق تابع را نیز داشته باشیم تا در هنگام BP از مشتق این تابع برای بروزرسانی وزن ها استفاده کنیم.
5. تابع فعال ساز لایه خروجی، تابع Softmax می باشد.
   1. از این قسمت با توجه BP بودن شبکه ما، به این نتیجه می رسیم که باید مشتق تابع را نیز داشته باشیم تا در هنگام BP از مشتق این تابع برای بروزرسانی وزن ها استفاده کنیم.
6. این شبکه را می خواهیم به کمک داده های Iris از کتابخانه sklearn آموزش دهیم.

از این قسمت به این نتیجه می رسیم که:

1. الگوریتم قرار است که به روش نظارت شده[[1]](#footnote-1) آموزش ببیند.
2. داده Iris باید مطابق با سوال 4 ورودی در ورودی داده هایش داشته باشد.
3. همچنین این مجموعه داده باید دارای 3 نوع خروجی باشد.

نکته ای که در مورد خروجی های این مسئله می باشد این است که هر سری خروجی ستایی ای که از شبکه گرفته می شود با یک بردار سه تایی به عنوان جواب های مطلوب مقایسه می شود که در این بردار تنها مقدار نرونی که مربوط به آن دسته ای می باشد که در ورودی مقادیرش دریافت شد، برابر با یک می باشد و دو دسته دیگر را برابر صفر قرار می دهیم تا بتوانیم نرون ها را آموزش دهیم.

* اصطلاحا نیز به این حالت one-hot-encoded می گویند به عبارت دیگر شکل مقابل و مطابق با آنچه که در این [لینک](https://towardsdatascience.com/deep-learning-which-loss-and-activation-functions-should-i-use-ac02f1c56aa8) مطالعه شد.

**مجموعه داده Iris**

در این سوال قرار است تا ما از مجموعه داده های Iris برای آموزش شبکه عصبی استفاده کنیم.

Iris یک مجموعه داده ای می باشد که به ما 4 تا عدد می دهد و خروجی مطلوب آن چهار عدد را نیز به ما می دهد. ما 4 عدد را به عنوان ورودی باید به الگوریتم دهیم، و الگوریتم به ما یک خروجی می دهد که ما باید با بروزرسانی وزن ها این خروجی را به خروجی مورد نظر برای آن اعداد، نزدیک کنیم.150 تا داده ورودی 4تایی داریم، همچنین 150 تا داده تکی خروجی داریم که نرونی که فعال می شود را مشخص می کند.

**الگوریتم Backpropagation**

این الگوریتم یک الگوریتم بهینه سازی محسوب می شود و به این شکل است که، یکبار با ورودی ها به سمت انتها الگوریتم حرکت می کنیم و خروجی را با وزن ها اولیه ای که تعیین کرده بودیم بدست می آوریم و بعد از بدست آوردن خروجی با مقایسه خروجی بدست آمده و خروجی مطلوب مورد نظرمان میزان خطا را بدست می آوریم و برای کمینه کردن این مقدار خطا به عقب بر می گردیم و مقدار وزن های الگوریتم را طوری بروزرسانی می کنیم که این میزان خطا را کاهش دهد.

پس بطور کلی در دو سمت حرکت می کنیم:

* 1. حرکت رو به جلو برای بدست آوردن خروجی شبکه

در این حرکت ما از توابع فعال ساز استفاده می کنیم تا ورودی لایه بعد و یا خروجی شبکه عصبی را بدست آوریم.

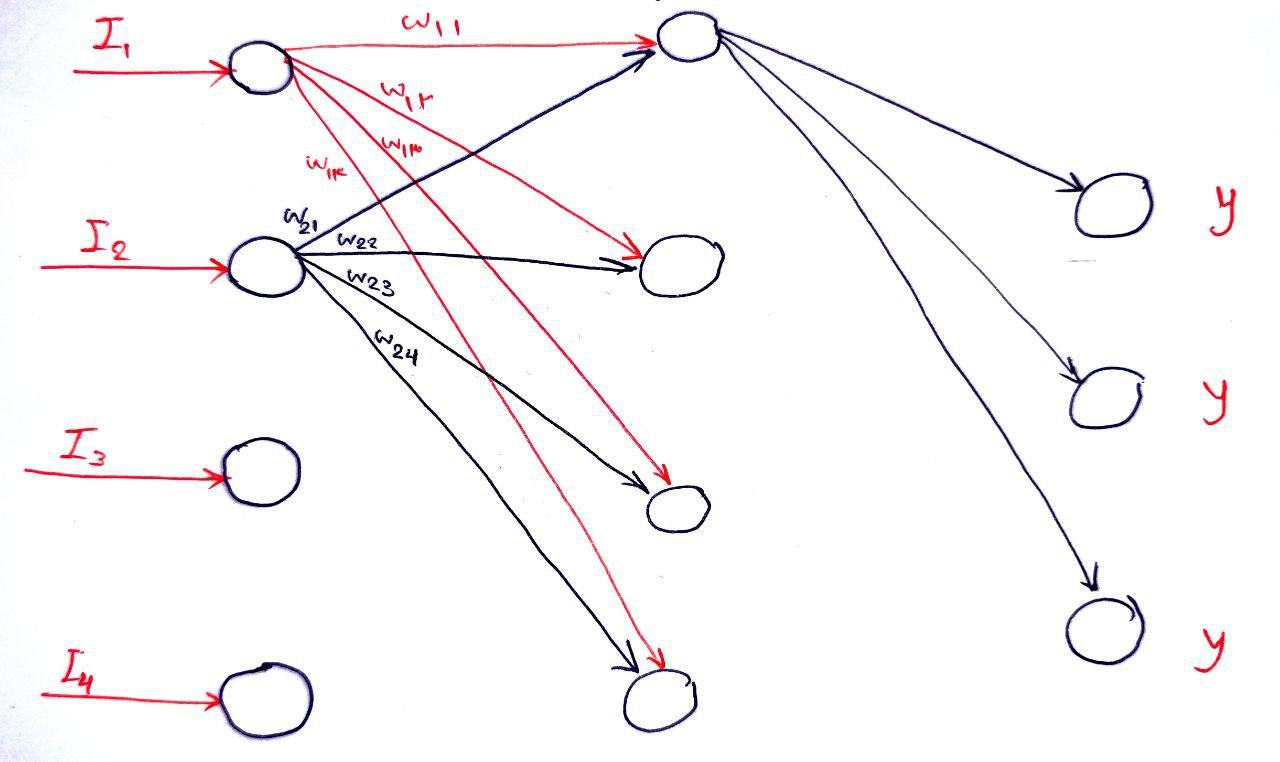
* 1. حرکت رو به عقب برای بروزرسانی وزن ها های شبکه عصبی

این حرکت که بعد از حرکت رو به جلو می باشد، برای بروزرسانی وزن ها می باشد و در این حرکت ما به مشتق توابع فعال ساز برای بروزرسانی وزن ها نیاز داریم.

**شبکه عصبی مسئله ما**

در این سوال همانطور که پیش تر بیان شده است، ما یک شبکه عصبی دو لایه داریم. در لایه ورودی 4 نرون داریم، در لایه پنهان نیز4 نرون داریم و در لایه خروجی 3 نرون داریم.

پس شبکه عصبی ما به شکل زیر خواهد بود برای این مسئله :



همانطور که از قبل می دانیم، تعداد وزن هایی که هر نرون در لایه قبل دارد برابر است با تعداد نرون هایی که در لایه بعد داریم؛ که در تصویر بالا نمونه آن را میبینید. در این تصویر تمامی وزن ها رسم نشده است و فقط وزن تعدادی از نرون ها برای واضح بودن شکل شبکه رسم شده است.

* نکته: یک ورودی ای که می تواند شبکه جدا از نرون های لایه قبل داشته باشد، Augmented input می باشد. که در این شبکه، ما از اضافه کردن این ورودی صرف نظر کرده ایم.

پس در حالت کلی بدون در نظر گرفتن ورودی ای دیگر، تعداد کل وزن هایی که وجود دارد در شبکه ما برابر است با، 4 نرون در لایه ورودی که هر یک 4 وزن دارند و 4 نرون در لایه پنهان که هرکدام 3 وزن دارند یعنی 4 \* 4 تا وزن برای انتقال از لایه ورودی به لایه پنهان و 4 \* 3 وزن برای انتقال از لایه پنهان به لایه خروجی و در حالت کلی تعداد وزن هایی که شبکه ما دارد و باید آموزش داده شود برابر است با 4 \* 4 + 3 \* 4.

در هنگام ایجاد شبکه عصبی ما وزن ها را به صورت رندوم مقدار دهی می کنیم. تنها دقت شود که مقدار هیچ وزنی نمی تواند برابر با صفر باشد. در اینجا ما برای مقدار دهی اولیه و رندوم وزن ها از تابع uniform random استفاده کرده ایم.

حرکت رو به جلو

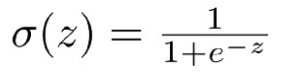
**در این قسمت می خواهیم، حرکت رو به جلو از سمت ورودی به سمت خروجی را توضیح دهیم. در حرکت رو به جلو این الگوریتم[[2]](#footnote-2) چندین مرحله ساده باید طی شود:**

1. **محاسبه مجموع حاصل ضرب وزن در ورودی اش که وارد یک نرون می شود. برای مثال به نرون اول لایه دوم 4 وزن وارد می شود که هر کدام از این وزن ها، وزن اول هر یک از نرون های لایه ورودی می باشد که این وزن ها در ورودی خودشان ضرب می شوند و نهایتا تمامی حاصل ها را با هم جمع میکنیم و در قالب یک عدد وارد نرون اول لایه دوم می شود و به همین ترتیب برای نرون های دیگر لایه دوم عمل می کنیم و پیش می رویم.**
2. **خروجی نرون های لایه دوم، در واقع از طریق تابع فعال ساز بدست می آید. اگر در مسئله تابع فعال سازی برای لایه ای ذکر شده باشد، خروجی ای که از این تابع فعال ساز می گیریم درواقع همان ورودی برای لایه بعدی می باشد. محاسبه این خروجی به این شکل است که همانطور که پیش تر گفتیم نهایتا یک عدد است که وارد هر یک از نرون های لایه دوم می شود. این عدد برای خارج شدن از این لایه و یا استفاده مجدد باید از تابع فعال ساز عبور کند. در واقع تابع فعال ساز میزان فعال شدن یک نرون را مشخص می کند، به همین دلیل اگر برای لایه ای تابع فعال سازی در نظر گرفته شده باشد، تنها خروجی های این تابع برای آن لایه است که مد نظر ما می باشد. در اینجا نیز این عددی که وارد هر یک از نرون های لایه دوم می شود باید از تابع فعال ساز این لایه که در این مسئله این تابع برای لایه پنهان تابع Sigmoid در نظر گرفته شده است عبور کند. پس خروجی این تابع است که خروجی لایه دوم را برای ما مشخص میکند، نه عددی که وارد نرون می شود.**
3. **خروجی نرون های لایه دوم ( که از طریق تابع فعال ساز بدست آورده شده است) به همین ترتیب ورودی برای نرون های لایه بعد می شوند که در وزن ها ضرب می شوند. و نهایتا در قالب یک عدد در این مسئله وارد نرون های لایه بعد (در اینجا لایه آخر) می شود. دقت شود که در این مسئله برای لایه خروجی نیز یک تابع فعال ساز در نظر گرفته شده است، پس همانطور که پیش تر ذکر شد ما باید اعدادی که وارد نرون ها لایه خروجی می شوند را از این تابع عبور دهیم تا این تابع عدد اصلی و مد نظرمان را به ما بدهد. در این مسئله عددی که این تابع یعنی تابع** Softmax **به ما می دهد خروجی های شبکه می باشد که باید با خروجی های مورد نظر ما مقایسه شوند که این مقایسه نیز به کمک تابع** crossentropy **صورت می پذیرد و نهایتا خطا آن محاسبه شود.**

تابع **Sigmoid**

**در این مسئله این تابع، بعنوان تابع فعال ساز لایه دوم(لایه پنهان) انتخاب شده است، از این تابع در هنگام حرکت رو به جلو استفاده می شود، و خروجی این تابع، خروجی لایه دوم و یا ورودی برای لایه سوم محسوب می شود.**

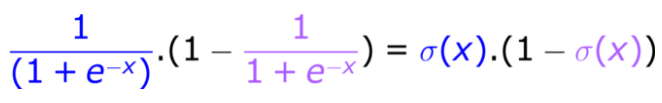
**این تابع به فرم زیر می باشد:**

****

**بعد از اینکه عددی که وارد هر یک از نرون های لایه دوم می شود، بدست آمد، باید آن عدد را از این تابع عبور دهیم تا عدد اصلی مورد نظرمان بدست آید.**

مشتق تابع **Sigmoid**

**مشتق این تابع در زمان حرکت به سمت عقب و بروزرسانی وزن ها مورد استفاده قرار می گیرد. که مشتق این تابع به فرم زیر می باشد:**

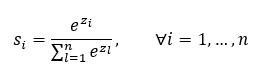
****

**در واقع برابر است با sigmoid(x) \* ( 1 – sigmoid(x)) .**

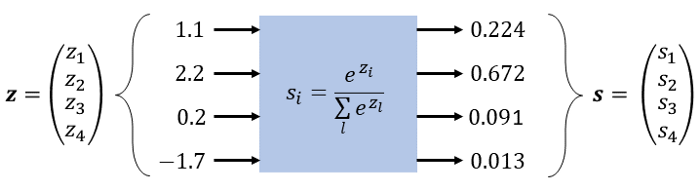
تابع **Softmax**

**تابع** Softmax **معمولا در لایه خروجی مورد استفاده قرار می گیرد که در این مسئله نیز این تابع، بعنوان تابع فعال ساز لایه خروجی انتخاب شده است، از این تابع در هنگام حرکت رو به جلو استفاده می شود. و خروجی این تابع خروجی شبکه ما محسوب می شود که نهایتا با خروجی مد نظر ما مقایسه می شود.**

**این تابع به فرم زیر می باشد:**

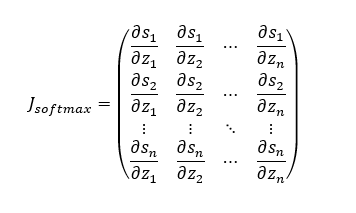
****

**در تصویر پایین یک مثال از طریقه استفاده شدن آن مشاهده می فرمایید:**

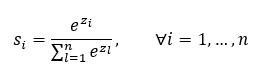
**برای اطلاعات بیشتر به این** [**لینک**](https://en.wikipedia.org/wiki/Softmax_function) **مراجعه کنید.**

مشتق تابع **Softmax**

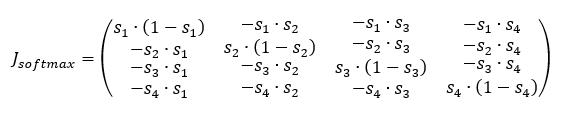
**زمانی که از مشتق تابع** softmax **صحبت می شود، در واقع ما داریم در مورد** Jacobian matrix صحبت می کنیم که ماتریسی به فرم پایین است:



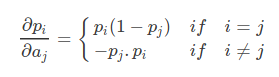
که در آن Si ها همان:



هستند، زیرا که در حالت کلی تابع Softmax تابعی است که یک بردار را در ورودی می گیرد و یک بردار دیگر را تحویل می دهد، در نتیجه در مشتق گرفتن ما با یک ماتریس طرف هستیم. در نتیجه ساده سازی این ماتریس ما به حالت کلی و فرمول زیر می رسیم:



**که میتوانیم فرمول زیر را از آن نتیجه بگیریم :**

****

**که در این فرمول Pi  همان Si می باشد که پیش تر در مورد آن گفتیم. برای اطلاعات بیشتر این** [**لینک**](https://towardsdatascience.com/derivative-of-the-softmax-function-and-the-categorical-cross-entropy-loss-ffceefc081d1) **را بررسی کنید.**

تابع **Cross Entropy**

**در حالت کلی وقتی که ما از شبکه یک خروجی بدست می آوریم، نوبت آن می رسد که خروجی بدست آورده شده را با خروجی مطلوبمان مقایسه کنیم و نهایتا در جهت کاهش اختلاف بین خروجی بدست آورده شده و خروجی مطلوب، حرکت کنیم. برای اینکار می توانیم از توابع مختلفی استفاده کنیم مانند** MSE **و یا** crossentropy**، که در این مسئله از آنجایی که از تابع** Softmax **استفاده می کنیم، که تابعی است که مقدار خروجی اش ما بین صفر و یک است، می خواهیم برای تابع هزینه از تابع** crossentropy **استفاده کنیم، که در مسائلی که از** Softmax **استفاده شده است تابعی مناسب می باشد. پس هدف ما کاهش مقدار این تابع است.**

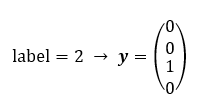
**قبل از هرچیز لازم به ذکر است که ما دو حالت مشخص برای رده بندی دسته ها داریم. به عبارت دیگر مسائل** classification **در نمونه هایشان به دو حالت تقسیم می شوند:**

**1. حالت اول به این شکل** که هر نمونه مختلف می تواند متعلق به چندین گروه **و یا دسته بندی باشد.**

**2. حالت دوم به این شکل** که هر نمونه مختلف می تواند تنها متعلق به یک گروه **یا دسته بندی باشد.**

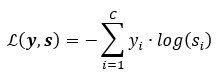
**حالا مسئله ما نیز حالت دوم را دارد. زیرا مجموعه داده های Iris که دارای ورودی هستند. هر ورودی تنها یک شی خاص را مشخص می کند و به این شکل نیست که یک ورودی بتواند متعلق به چندین دسته باشد، و اگر خروجی های مطلوب ورودی را نیز بررسی کنید، مشاهده می کنید که در حالت کلی ما 3 نوع خروجی داریم، خروجی صفر، یک و دو که هر یک، یک دسته خاص را مشخص می کند به همین دلیل است در واقع تابع softmax بهترین تابع می تواند باشد در خروجی تا احتمال تعلق گیری یک دسته را مشخص کند.**

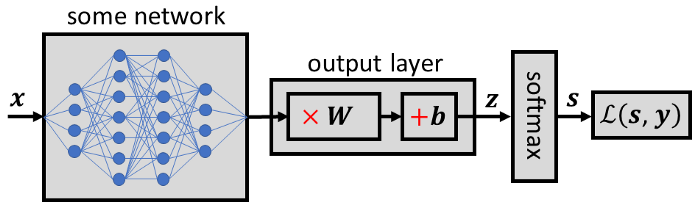
**پس در نتیجه برای این مسئله ما با حالت دوم طرف هستیم، یعنی حالتی که هر ورودی تنها می تواند یک دسته را مشخص کند. تابع هزینه**cross entropy **تابعی است که در حالت های نوع دوم مورد استفاده قرار می گیرد، و درواقع این تابع multi-class classification یعنی مشخص می کند که هر نمونه دقیقا برای کدام یک از دسته های می باشد و اینکار با دادن مقدار یک به دسته ای که آن نمونه به آن تعلق دارد و مقدار صفر به باقیه دسته ها انجام می دهد(اصطلاحا One hot encoded). مانند تصویر زیر :**

****

**برای اطلاعات بیشتر این** [**لینک**](https://towardsdatascience.com/derivative-of-the-softmax-function-and-the-categorical-cross-entropy-loss-ffceefc081d1) **را بررسی بفرمایید.**

**تابع** crossentropy **به شکل مقابل می باشد:**

****

**در این تصویر** y **درواقع همان خروجی مطلوب و مد نظر ما می باشد و** S**iخروجی ای می باشد که ما از شبکه گرفتیم. هزینه برای هر نرون به کمک همین فرمول محاسبه می شود. همچنین c برابر است با تعداد دسته بندی هایی که مسئله ما دارد که در اینجا برابر با 3 می باشد(به عبارت دیگر برابر است با تعداد نرون های لایه خروجی).**

**در نتیجه در اینجا** y **را که در مجموعه داده** Iris **داریم و** Si **هم همان خروجی های هر نرون در لایه آخر می باشد که به کمکSoftmax بدست آمده است. Z همان مقادیری است که از ضرب ورودی در وزن و جمع حاصل آنها بدست آمده است و b همان بایاس ما می باشد که ما در این مسئله آن را در نظر نگرفتیم.**

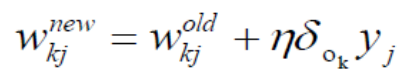
**برای حرکت به سمت نقاط مینیمم تابع هزینه، می توانیم از قواعد گرادیان استفاده کنیم. همانطور که در این لینک مشاهده می فرمایید، گرادیان تابع هزینه برابر است با:**

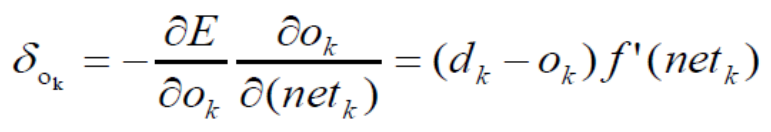
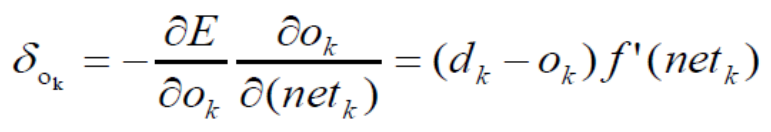
****

متد بروزرسانی وزن ها

**وزن جدید برابر است با وزن قدیم + { نرخ یادگیری \* اختلاف بین خروجی مطلوب و خروجی بدست آمده \* مشتق تابع فعال ساز آن لایه که در ورودی اش همان ورودی خودش در زمان فعال سازی را می گیرد(اگر تابع فعال ساز نداشتیم یک در نظر می گیریم) \* ورودی برای آن وزن که قبل تر استفاده شد(همان ورودی ای که از خروجی تابع فعال ساز لایه دوم یعنی سیگموید بدست آمد )}**

**یعنی تصویر پایین:**

****

**که در آن δ برابر است با :**

روند کلی الگوریتم

**تا به اینجای کار با بخش های مختلف الگوریتم بطور جداگانه و تقریبا مستقل آشنا شدیم، حالا می خواهیم یکبار مراحل را از اول پیاده سازی تا انتها در اینجا بیان کنیم تا پیچیدگی مسئله را کم کنیم. در اینجا سعی می کنیم از گفتن نکات تکراری و جزییات غیر ضروری پرهیز کنیم.**

* حرکت رو به جلو

1. **ایجاد شبکه و مقدار اولیه وزن ها**

**در مرحله اول ما باید یک شبکه مطابق با تعداد نرون و لایه های خواسته شده صورت مسئله ایجاد کنیم. در این مرحله باید وزن هارا خودمان بصورت رندوم مقدار دهی کنیم و باید مراقب این نیز باشیم که هیچ وزنی مقدار صفر به خود نگیرد.**

**مطابق با صورت مسئله و آنچه پیش تر ذکر شد، یک شبکه با 4 ورودی، 4 نرون در یک لایه پنهان و 3 نرون در لایه خروجی را ایجاد می کنیم.**

1. **انتقال داده از ورودی به لایه پنهان**

**در این مرحله باید هر ورودی در وزن هایش ضرب شود و وارد نرون خودش در لایه پنهان شود.(همانند آنچه پیش تر ذکر شد)**

1. **جمع کردن مقادیر ورودی به نرون های لایه پنهان**

**تمام مقادیر حاصل از ضرب ورودی در وزن نرون را با هم دیگر جمع می کنیم و یک مقدار ایجاد می کنیم.**

1. **فعال سازی نرون های لایه پنهان به کمک تابع** Sigmoid

**تابع فعال سازی لایه پنهان را پیاده سازی می کنیم. و مقداری که در مرحله قبل برای هر یک از نرون های لایه پنهان بدست آوردیم را از تابع فعال ساز عبور می دهیم تا میزان فعال شدن هر نرون مطابق با تابع فعال ساز مشخص شود، و یک مقدار جدید بدست آوریم.**

1. **انتقال داده از لایه پنهان به لایه خروجی**

**مقدار حاصل از تابع فعال ساز لایه پنهان، خروجی نرون های لایه پنهان و در واقع ورودی برای نرون های لایه آخر می باشد. این ورودی ها را در وزن هایش ضرب می شوند و وارد نرون خودش در لایه آخر می شود.**

1. **جمع کردن مقادیر ورودی به نرون های لایه آخر**

**تمام مقادیر حاصل از ضرب ورودی در وزن نرون را با هم دیگر جمع می کنیم و یک مقدار ایجاد می کنیم.**

1. **فعال سازی نرون های لایه خروجی به کمک تابع** Softmax

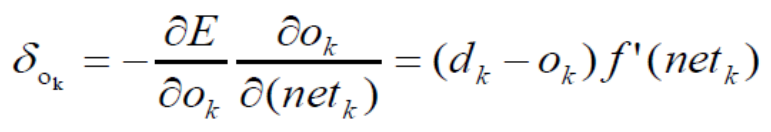
**تابع فعال سازی لایه خروجی را نیز پیاده سازی می کنیم. و مقداری که در مرحله قبل برای هر یک از نرون های لایه خروجی بدست آوردیم را از تابع فعال ساز عبور می دهیم تا میزان فعال شدن هر نرون مطابق با تابع فعال ساز مشخص شود، و یک مقدار جدید بدست آوریم.**

* + **این مقدار بدست آمده خروجی شبکه می باشد، که باید با خروجی مطلوب برای آن ورودی مقایسه شود.**
* حرکت رو به عقب

1. **محاسبه دلتا لایه آخر (البته ما در کدمان مستقیما از گرادیان استفاده کردیم.)**

**محاسبه اختلاف بین خروجی بدست آمده \* مشتق تابع فعال ساز آن لایه با ورودی های وزن دار شده قبلی(اگر آن لایه تابع فعال ساز نداشت این قسمت برابر با یک خواهد بود) که در واقع این بخش محاسبه همان δ می باشد، که برای هر نرون یک عدد است.**

**دلتا لایه آخر به کمک فرمول زیر محاسبه می شود:**

****

**که همانطور که گفتیم:**

**dk خروجی مطلوب ما می باشد.**

**Ok خروجی ای می باشد که از شبکه بدست آوردیم.**

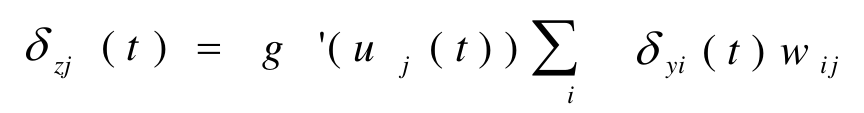
**F`مشتق تابع فعال ساز لایه آخر می باشد، در صورتی که تابع فعال سازی نداریم برای لایه آخر مقدار آن یک فرض می شود.**

**Netk ورودی های وزن دار شده (ورودی های موثری) ای می باشند، که وارد نرون شده اند(قبل از اینکه از تابع فعال ساز عبور کنند)**

1. **محاسبه دلتا لایه پنهان (البته ما در کدمان مستقیما از گرادیان استفاده کردیم.)**

**بعد از محاسبه دلتا برای لایه آخر باید دلتا لایه های پنهان را نیز محاسبه کنیم. هدف ما در الگوریتم BP نیز دقیقا همین بوده است، ما چون خروجی های مطلوب لایه های پنهان را نداریم، مجبور می شویم یکبار تا آخر شبکه پیش برویم و خروجی های شبکه را حساب کنیم و سپس به کمک گرادیان بر گردیم و خروجی های مطلوب لایه پنهان را نیز محاسبه کنیم تا بتوانیم به کمک آن وزن ها را به درستی و مطابق چیزی که در نظر داریم بروز رسانی کنیم.**

**دلتا لایه پنهان از فرمول زیر محاسبه می شود:**

****

**G` مشتق تابع فعال ساز این لایه**

**Uj(t) ورودی های وزن دار شده (ورودی های موثر) ای می باشند، که وارد نرون شده اند (قبل از اینکه از تابع فعال ساز عبور کنند).**

Yi(t) \* Wij**δ در واقع همان دلتاهایی می باشد که در لایه قبل بدست آمده اند و وزن دار شده اند و ما در اینجا مجموع تمام این دلتا هایی که وارد نرون لایه پنهان می شوند از در برگش را میخواهیم.**

* **توجه شود که در اینجا قبل از اینکه وزن های لایه آخر را بروزرسانی کنیم، دلتا های لایه پنهان را بدست می آوریم، و سپس بعد از آن سراغ بروز رسانی وزن ها می رویم.**

1. **بروز رسانی وزن های لایه آخر**

**سپس وزن های لایه آخر را متناسب با دلتا های بدست آمده بروزرسانی می کنیم.**

1. **بروزرسانی وزن های لایه پنهان**

**سپس وزن های لایه پنهان را متناسب با دلتا های بدست آمده بروزرسانی می کنیم.**

1. **یک مرحله epoch برای الگوریتم تمام می شود، و داده های ورودی مان آموزش داده می شوند، و در ادامه مجددا خروجی ها و ورودی های بعدی بررسی و وزن هارا بروز رسانی می کنیم.**

**لازم به ذکر است که توضیحاتی دیگر نیز در فایل Jupyter نوشته شده است که خواندن آن و بررسی آن خالی از لطف نیست. همچنین به دلیل اروری نا مشخص برای لود شدن داده های iris که در jupyter نوشته شده است، نمیشود که برنامه را تحت jupyter اجرا کرد و تنها باید از فایل کد با فرمت .py برای اجرای برنامه اقدام کنید و در فایل ژوپیتر را تنها یکسری از توضیحات دیگر نوشته شده (البته به زبان انگلیسی)**

**و همچنین در نوشتن کد برنامه سعی شد تا برای تمامی متد ها و یا کلاس ها داکیومنتیشن و کامنت های لازم و به شکلی واضح کننده قرار داده شود تا نه تنها خوانایی کد بالاتر برود بلکه مراحل انجام دادن بطور کامل مشخص باشد حتما حتما آن داکیومنت و کامنت ها نیز ملاحظه شود.**

**برای بررسی کد کافیست تا فایل کد را مشاهده بفرمایید، سعی شد در پیاده سازی از توضیحات کافی و اسامی مناسب برای متغیر استفاده شود.**

سوال دوم – دسته بندی مجموعه داده **MNIST** در **MLP**

**قرار است تا به کمک ابزار هایی که پایتون در اختیار ما قرار داده است، این سوال را حل کنیم. در این بخش قصد داریم تا این سوال را به کمک یک شبکه** MLP **با دو لایه پنهان به عرض 300 و یک لایه خروجی به عرض 10 حل کنیم.**

**بررسی کد پیاده سازی شده:**

1. **دانلود مجموعه داده**

**قبل از هرچیز ما مجموعه داده های عددی** MNIST **را دانلود کردیم و همچنین در کنار آن داده های تست را نیز دانلود کردم تا نهایتا خروجی را در قالب نمودا بتوانیم با هم بررسی کنیم.**

**تصویر آدرس داده ها ................................**

1. **مشخص سازی نوع دسترسی و تغییر هر داده در کد**

**در اینجا سعی کردیم تا داده ها را با تبدیل به تنسور مورد استفاده قرار دهیم، زیرا که** Transforms.ToTensor() **برای تصاویر مورد استفاده قرار می گیرد به این دلیل که هر یک از المنت های تصاویر در بازه 0 تا 255 می باشد.**

1. **وقتی هر عکس را بررسی میکنیم مشاهده میکنیم که ابعاد هر عکس 28 پیکسل در 28 پیکسل است. و آن عدد یکی که مشاهده می کنید درواقع تعداد کانال های تصویر می باشد.**

بررسی **Multilayer Perceptron** در مجموعه داده های **MNIST**

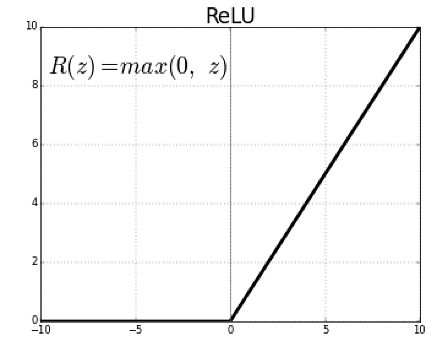
1. **یک شبکه چند لایه تشکیل شده از تعدادی لایه شامل نرون می باشد.**
2. **هر یک از نرون های این لایه های هوشمند هستند، اما تنها میتوانند کار های بسیار بنیادی و کوچکی را انجام دهند. (مانند نرون های مغزمان)**
3. **کاری که در هر یک از نرون ها میتواند انجام گیرد، در واقع این رابطه N = Wx + b می باشد. که با پارامتر های آن به خوبی آشنا هستیم.**
4. **هر نرون در هر لایه به تمام نرون های لایه بعدی خود متصل است.**
5. **در شبکه های چند لایه پرسپترونی داده ها رو به جلو حرکت می کنند و به** Feed-Forward Networks معروف هستند.
6. سه مولفه یا لایه در شبکه داریم:
7. لایه ورودی **(یک لایه است در تمام حالات)**
8. ورودی داده ها از این لایه وارد می شود.
9. لایه خروجی **(یک لایه است در تمام حالات)**
10. **خروجی شبکه ما از این لایه بدست می آید و مسئله** classification/regression به گونه ای در این لایه مشخص و بررسی می شود.
11. در این مسئله ما 10 کلاس خروجی داریم که هر یک ارقام 0 تا 9 را مشخص می کنند. به همین دلیل است که ما در لایه خروجی 10 نرون داریم، تا بتوانیم این کلاس هارا از یک دیگر جدا کنیم.
12. **لایه های پنهان(می تواند بیش از یک لایه باشد در تمام حالات)**
13. لایه هایی که مابین لایه ورودی و لایه خروجی قرار دارند لایه های پنهان تلقی می شوند که آخرین لایه پنهان ورودی های لایه خروجی را مشخص می کند.
14. تصاویر با پیکسل های 28 در 28

**پیکسل های 28 \* 28 هر تصویر برای این که به یک بردار یک بعدی تبدیل شوند** FLATTENED **شده اند، که در نتیجه آن ما 784 مقدار پیکسل داریم که باید از ورودی دریافت کنیم برای این مسئله.**

**پس یکی از کار هایی که انجام می دهیم تبدیل این بردار دو بعدی (28 در 28) به یک بردار یک بعدی که شامل 784 مقدار می شود می باشد.**

1. **تنها کاری که در لایه ورودی باید انجام دهیم ارسال این 784 مقدار به اولین لایه پنهان می باشد.**
2. **اولین محاسبات ما در اولین لایه مخفی شروع میشود. مطابق با صورت سوال برای این لایه 300 نرون در نظر گرفته شده است که هر یک از ورودی های وارد آن می شوند. هر یک از نرون های اولین لایه مخفی یک خروجی تولید می کنند که این خروجی ورودی ای برای لایه دوم مخفی که آن هم دارای 300 نرون است می باشد. البته لازم به ذکر است که خروجی هر نرون از لایه اول مخفی باید از تابع فعال ساز خودش عبور کند و خروجی تابع فعال ساز است که ورودی برای نرون های لایه بعدی می باشد.**
3. **توابع فعال ساز به ما کمک می کنند تا بتوانیم خروجی را در یک بازه محدود بیاوریم و فعال سازی هر نرون را در آن بازه مورد بررسی قرار دهیم.**

**توابعی که میتوانستیم استفاده کنیم در اینجا، Sigmoid و یا Tanh و یا Relu و یا هر تابع فعال ساز دیگری بود، که ما در اینجا از ReLU استفاده کردیم.**

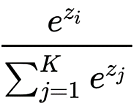
****

**از فواید و دلایل استفاده از این تابع این است که، این تابع مقادیر منفی را نادیده می گیرد و مقادیر مثبت را نگهداری می کند.**

1. **تمامی این اتفاقات در لایه مخفی دوم که آن نیز شامل 300 نرون است نیز رخ می دهد، و نهایتا خروجی آخرین لایه مخفی ورودی برای لایه خروجی می باشد.**
2. لایه خروجی

**همانطور که پیش تر نیز ذکر شد از آنجایی که ما با یک مسئله شامل 10 نوع کلاس طرف هستیم، پس طبیعی است که در لایه خروجی،10 نرون داشته باشیم.**

**همچنین برای لایه خروجی ما تابع** SoftMax **را به عنوان تابع فعال ساز انتخاب کرده ایم، تا بر اساس احتمال که به ما می دهد بتوانیم کلاسی که با احتمال بیشتری ممکن است داده به آن تعلق داشته باشد را انتخاب کنیم.**

**این تابع مطابق 10 نرونی که داریم، 10 خروجی به ما می دهد که مجموع این 10 عدد برابر یک می باشد.**

1. **سپس بعد از اینکه خروجی بدست آمد، خروجی را برای بروزرسانی وزن ها به عقب ارسال می کنیم. مطابق با آنچه در تمرین اول پیاده سازی کردیم.**
2. **در این سوال سعی کردیم که مقادیر ورودی داده هارا تکه تکه کنیم و ارسال کنیم به عبارتی به شکل semi-batch ارسال کردیم.**

**و بعد از هر batch که ارسال کردیم، خروجی را با خروجی مطلوب مقایسه می کنیم و loss محاسبه می شود.**

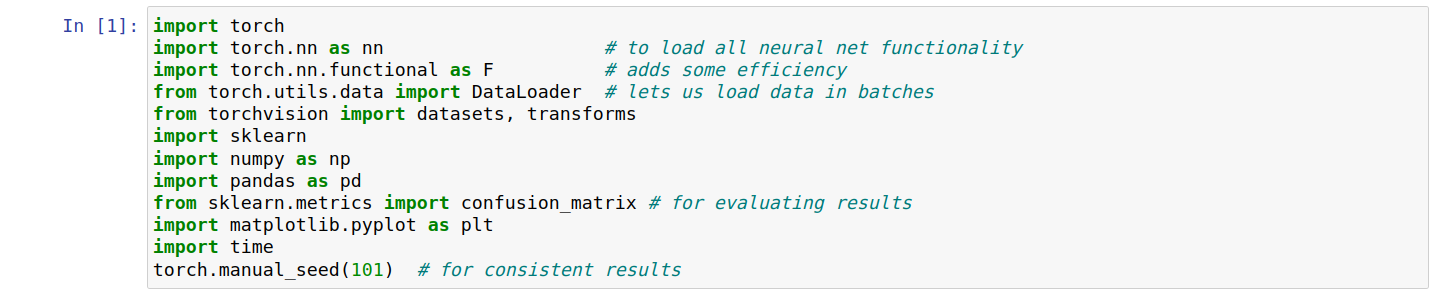
1. **همانطور که گفتیم ما کل داده هارا یک جا ارسال نمی کنیم و یا یکی یکی ارسال نمیکنیم بلکه سعی میکنیم با حالتی ما بین این دو پیش ببریم. از انجایی که 60000 هزارتا داده داریم ما در اینجا سعی کردیم که 100 بار مجموعه داده های 600 تایی را ارسال کنیم. پس در واقع ما در هر batch 600 تا داده را ارسال می کنیم.**
2. **می دانیم که برای مرحله آموزش، ارسال تعداد کمتری از داده ها باعث می شود که ما به عبارتی به لوکال مینیمم یا به عبارت دیگر بسیار به داده هایمان فیت شویم که در نتیجه آن ممکن است شبکه ما نسبت به داده های پرت و یا جدید خطای زیادی داشته باشد. حالا ما سعی کردیم در اینجا 100 سری داده 600 تایی ایجاد و بررسی کنیم.**
3. **در تست هم چون که آموزشی صورت نمیگیرد میتوانیم هر تعداد که میخواهیم را ارسال و بررسی کنیم.**
4. **یک عبارتی در زمان استفاده از داده ها وجود دارد به نام shuffle که کارش این است که بعد از هر epoch در مجموعه داده ما یکسری جابه جایی هایی بطور اتفاقی صورت دهد.**
5. **در این تمرین از ابزار های pytorch استفاده میکنیم در نتیجه ما پیاده سازی سختی نخواهیم داشت.**

**کافیست ورودی ها تعداد نرون و لایه ها را مشخص کنیم.**

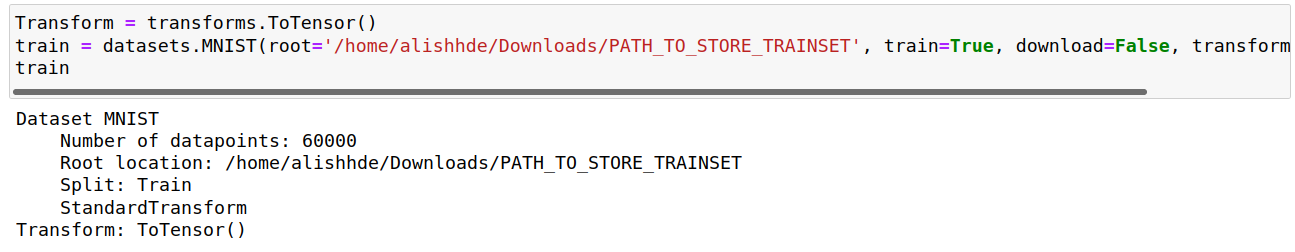
**حرکت رو به جلو و عقب برای آموزش را مشخص کنیم.**

**در ادامه توابع را مشخص کنیم و تابع loss را مشخص و تنظیم کنیم.**

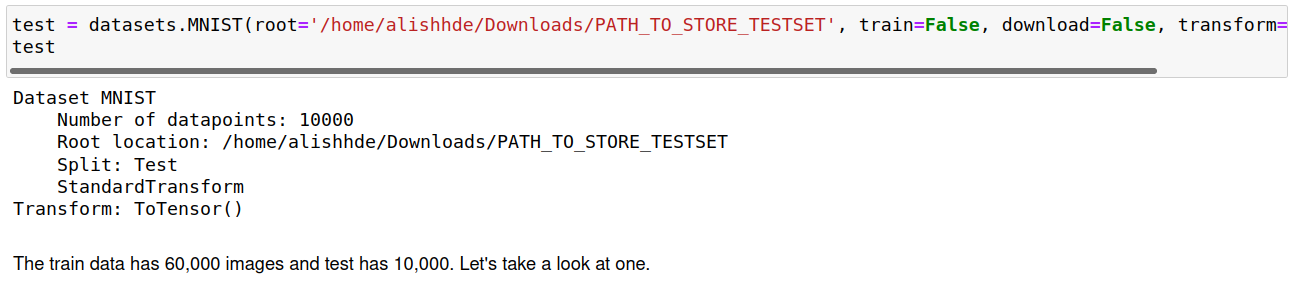
در تصویر پایین سعی کردیم تمامی کتابخانه های مورد نیاز خودمان را به برنامه اضافه کنیم.

****

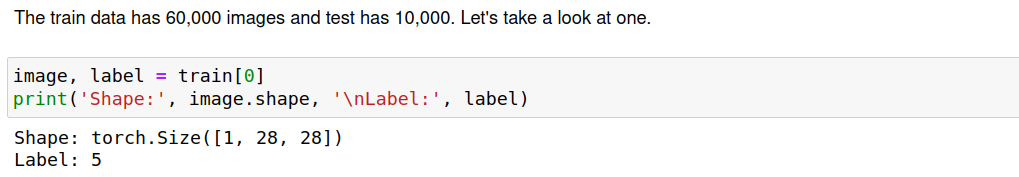
**در تصویر بعدی مقدار transform را به Transform که از نوع ToTensor() است مقدار دهی کرده ایم و همچنین از دو دسته داده ای که دانلود کردیم، داده آموزش را که حاوی 60000 هزارتا داده است را مشخص کرده ایم. دلیل اینکه download = False است این است که ما چون داده هایمان را از پیش دانلود کردیم و در لپتاپ خودمان داشتیم دیگر نیازی به دانلود مجدد آن نداشتیم و تنها آدرس داده ها را مشخص کردیم.**

****

**همچنین در تصویر پایین داده تست را نیز مشخص کردیم برای ارزیابی بیشتر. داده تست ما حاوی 10000 هزارتا داده می باشد همونطور که مشاهده می کنید.**

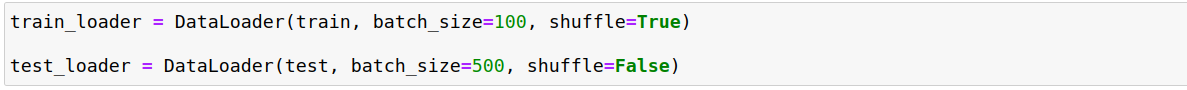
****

**همانطور که پیش تر نیز ذکر شد، تصاویر ما دارای ابعادی 28 در 28 پیکسل می باشند. برای مثال اولین داده ما عدد 5 می باشد، که در مجموعه داده ها با label 5 ذخیره شده و ابعادش هم 28 \* 28 می باشد.**

****

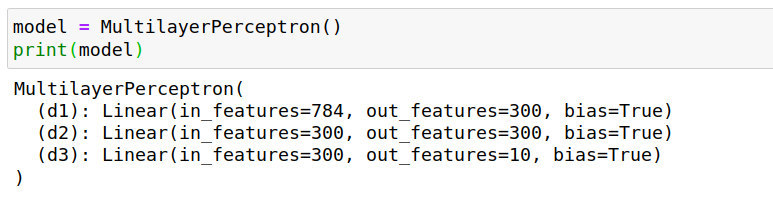
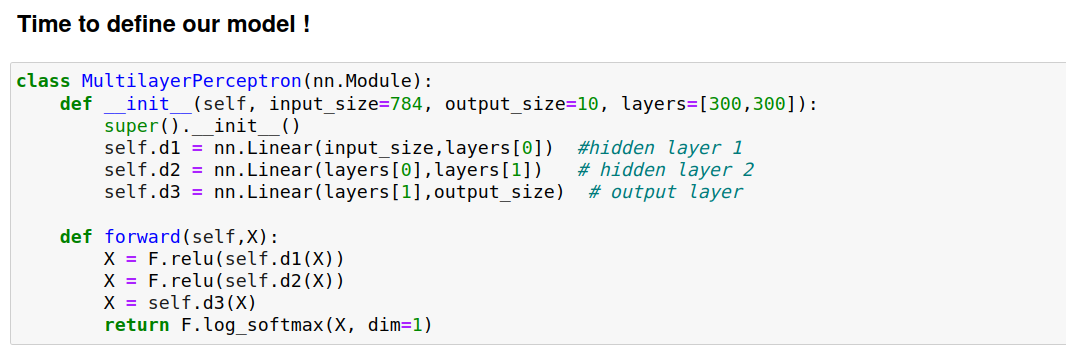
**سپس همانطور که پیشتر در مورد آن بحث شد، batch\_size خودمان را برابر 100 قرار دادیم و با توجه به 60000 تایی بودن داده های ما، در هر دور 600 داده ارسال می شود. همچنین batch\_size داده های تستمان را برابر 500 قرار دادیم.**

**دقت شود که میتوانیم این مقدار را در حالت های مختلف تغییر دهیم. و بررسی نماییم.**

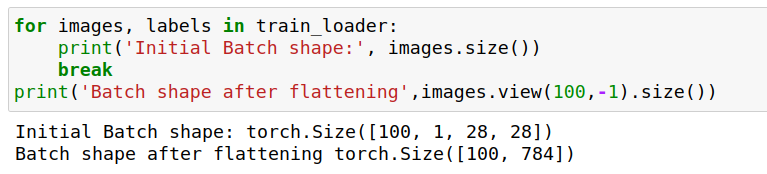
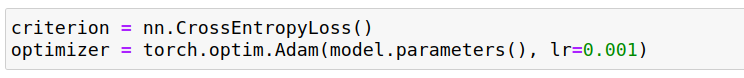
****

**در ادامه یک class از MLP خودمان تشکیل دادیم و در آن تعداد لایه های مخفی و تعداد ورودی و لایه خروجی را به همراه نرون های آن مطابق با صورت سوال مشخص کردیم. ورودی ما بدلیل اینکه هر تصویر 28 پیکسل در 28 پیکسل است و چون دو بعدی است، وقتی که میخواهیم این تصویر را یک بعدی کنیم در واقع تعداد مقادیر یک تصویر 28\*28 در یک ماتریس 28 در 28 هست که معادل آن در یک بردار یک بعدی برابر است با حاصل ضرب ابعاد در هم در نتیجه ما 784 ورودی داریم.**

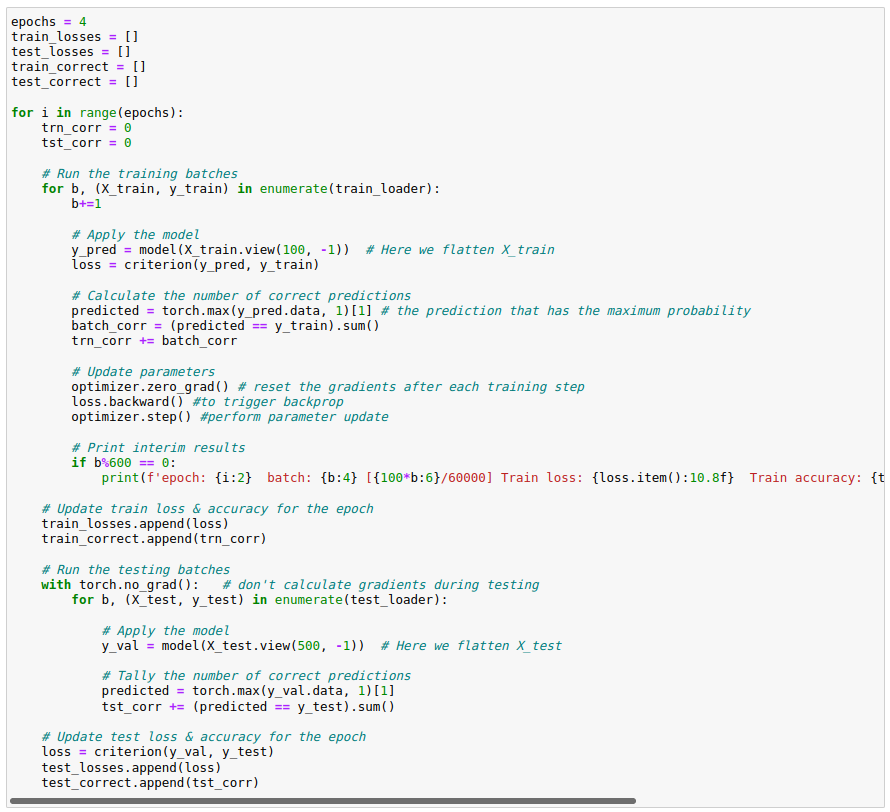
**همچنین در این کلاس متد حرکت رو به جلو را نیز تعریف کردیم، توابع فعال سازه لایه پنهان، تابع ReLU در نظر گرفته شده است، به همان دلیلی که پیش تر ذکر شد و تابع فعال ساز لایه خروجی نیز Softmax در نظر گرفته شد.**

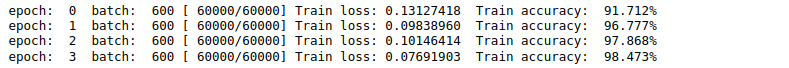
****

**تابع Crossentropy را بعنوان تابع خطا در نظر گرفتیم.**

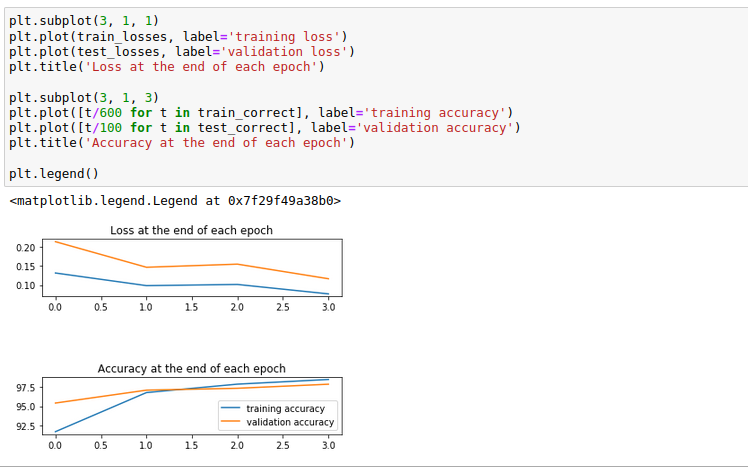
****

**در این تصویر تعداد epoch ها را برابر 4 قرار دادیم. و نهایتا به آموزش و بروز رسانی وزن و پیدا کردن خطا پرداختیم که در کامنت ها هر یک به درستی و کامل نشان داده شده است.**

****

**خروجی این کد را در ادامه مشاهده می فرمایید:**

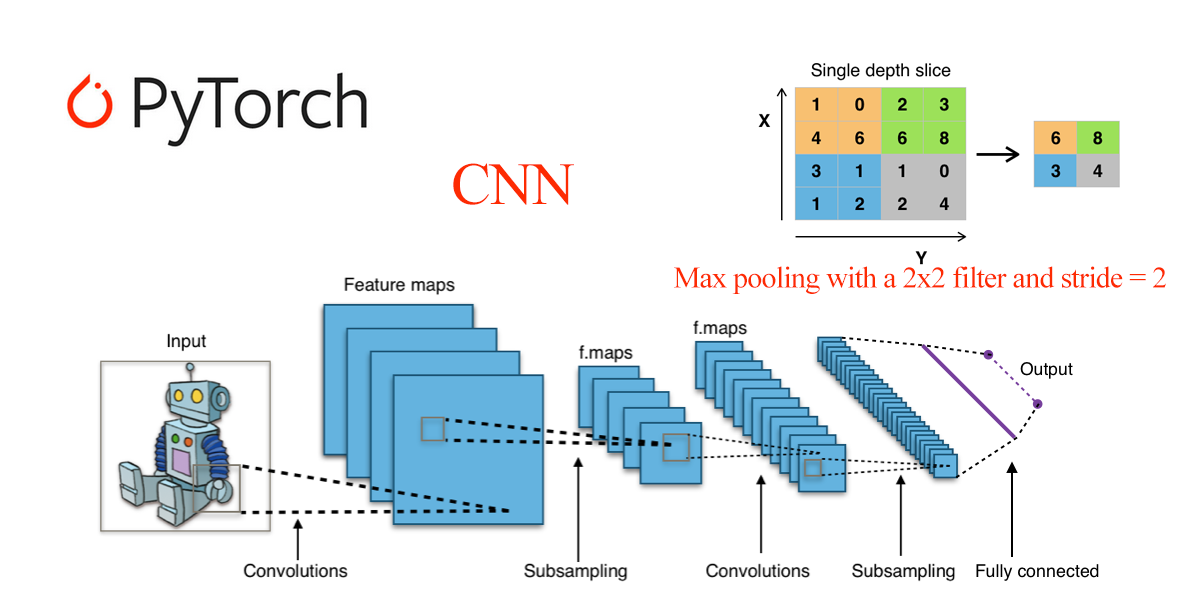
**و نهایتا خروجی تابع را مشاهده می فرمایید.میتوان مشاهده کرد که خروجی های تست و آموزش خیلی با هم فیت نیستند، اما اگر میخواهیم که مقدار فیت بودن آ« را کمتر کنم کافیست که چنتا از نرون هارا دراپ کنیم.**

****

سوال سوم – دسته بندی مجموعه داده **MNIST** در **CNN**

**CNN ها در واقع همان شبکه های عصبی می باشند با این تفاوت که در این سری از شبکه ها برای بررسی تصاویر از روابط کانوولوشنی استفاده شده است.**

**بطور کلی در این قسمت از سوال با فرایند زیر مواجه هستیم**

****

**از pytorch قدم به قدم برای پیاده سازی شبکه CNN استفاده می کنیم. و نهایتا مدلمان را با داده هایمان آموزش می دهیم و سپس ارزیابی می کنیم.**

**داده های MNIST را همانطور برای سوال دوم مجبور شدیم بطور جداگانه دانلود کنیم، برای این سوال نیز از همان فایل و داده هایی که در آنجا دانلود کردیم استفاده می کنیم. پس داده هارا با دستورات مورد نیازش در کد لود می کنیم.**

**مراحل کار در این سری از پیاده سازی در عکس کاملا مشخص شده است. از آنجایی که بیشتر مراحل کار مطابق سوال های قبلی می باشد از بازگویی یکسری از چیز خودداری می کنیم.**

**در این پیاده سازی ما از دو لایه کانوولوشنی استفاده می کنیم، همچنین از تابع ReLU به عنوان تابع فعال ساز استفاده می کنیم.**

**در ادامه به بررسی پارامتر هایی که در کد ایجاد کردیم و ممکن است نام و کارایی آن مشخص نباشد می پردازیم:**

1. **In\_channels**

**این متغیر تعداد کانال هایی که در ورودی تصویر داریم را مشخص می کند.**

1. **Out\_channels**

**این متغیر قرار است تا تعداد کانال هایی که توسط کانوولوشن ایجاد شده است را نگه داری کند.**

1. **Kernel\_size**

**این متغیر اندازه کرنل کانوولوشن گیری را مشخص میکند.**

1. **Stride**

**همانطور که می دانیم، stride میزان حرکت فیلتر را مشخص می کند که معمولا بطور پیش فرض برابر است با یک.**

1. **Dilation**

**فاصله بین المنت های کرنل می باشد.**

**در ادامه متد فوروارد را در کد قرار دادیم که با فرایند کار آن به خوبی در سوال های قبلی آشنا شدیم. سپس loss function را تعریف کردیم. و بعد از اتمام ایجاد شبکه، فرایند آموزش را از شروع کردم و داده های کان را به شبکه آموزش دادیم.**

**اطلاعات بیشتر را در کد که در ژوپیتر قرار دارد مشاهده می فرمایید. بدلیل کمبود وقت فرصتی برای قرار دادن تصاویر آن در اینجا وجود نداشت.**

**لازم به ذکر است که سوال دوم و سوم کدش به کمک گوگل پیاده سازی شد به دلیل کمبود وقت اما سعی کردم در مقابل با توضیح دادن و گزارش نویسی و تحلیل درست تر و کامل کد بیان کنم که، در نوشتن کد آن مشکلی برام وجود ندارد در صورت داشتن وقت.**

سپاس از وقتی که گذاشتید.

1. Supervised learning [↑](#footnote-ref-1)
2. Forward [↑](#footnote-ref-2)