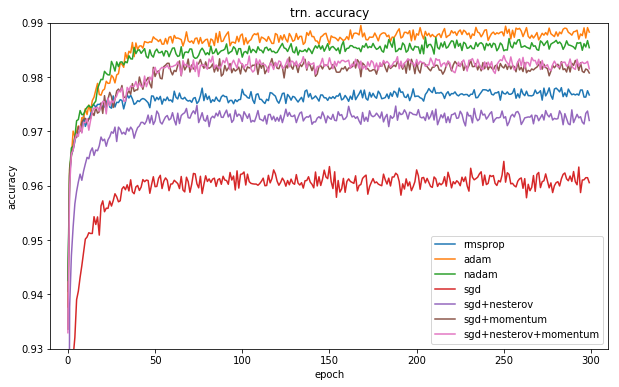
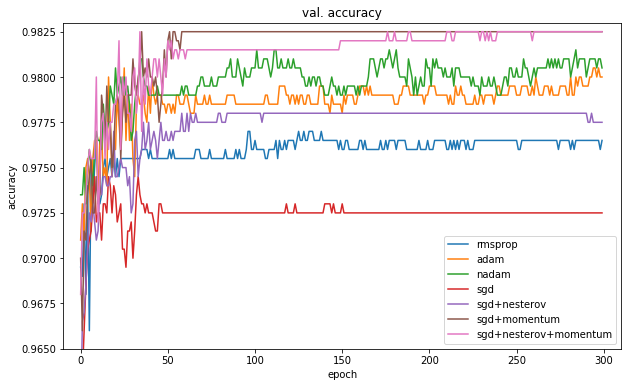
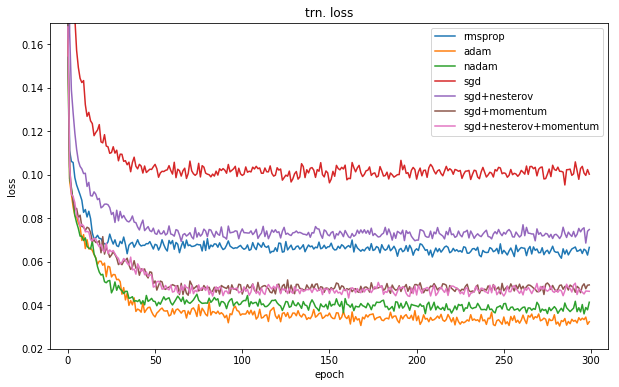
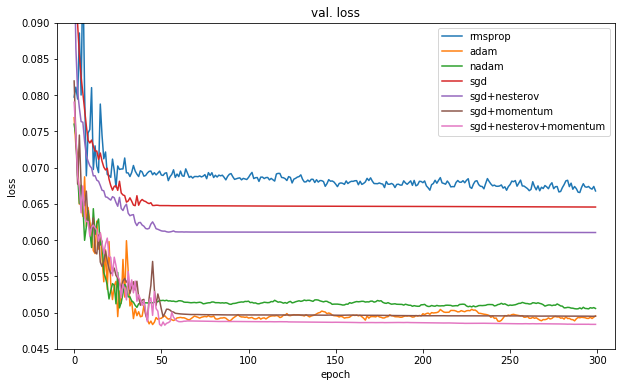
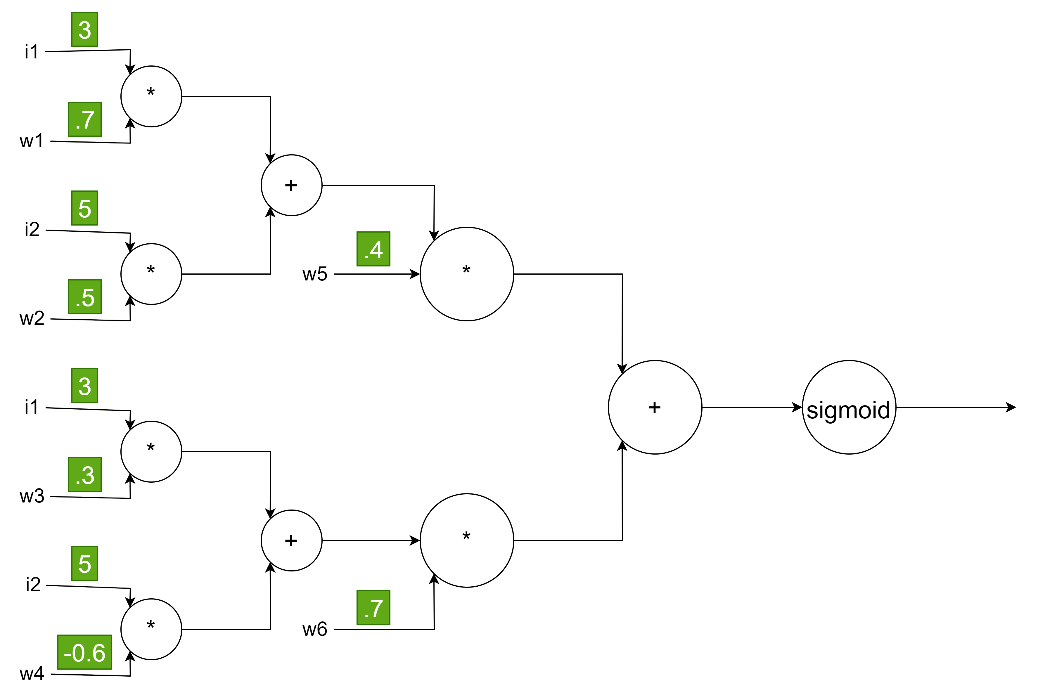
**امیرحسین احمدی 97522292 – تمرین چهارم**

1. در حالت کلی adam و کلا optimizer های adaptive خیلی سریع تر از sgd خالی همگرا میشوند و برای استفاده در شبکه های عصبی پیچینده و عمیق مناسب هستند. ولی با ازمایشات انجام شده، مشاهده میشود که عملکرد sgd (منظور نسخه های مختلف sgd است) در دیتای ولیدیشن بهتر از حتی adam است و در بسیاری از مقالات معروف از sgd به عنوان optimizer اصلی استفاده میشود ولی در دیتای ترین همچنان adaptive optimizer ها عملکرد بهتری دارند. نمونه ای از آزمایش انجام شده روی یک دیتاست(سمت راست برروی ولیدیشن و چپ بر روی ترین است):

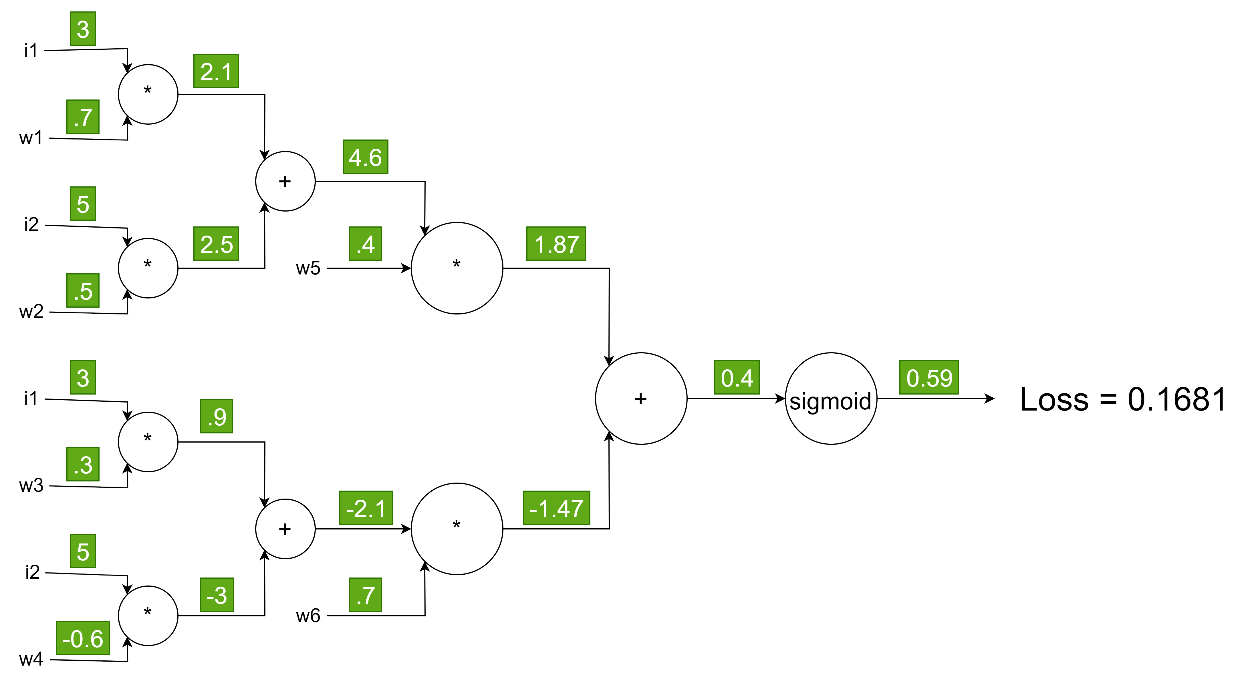




1. در ابتدا شبکه را به صورت زیر باز کرده (هر نورون را به ضرب و جمع تجزیه کرده به غیر از تابع فعال ساز نورون آخر که بصورت مستقیم آن را حساب کردم) تا محاسبات راحت تر باشد. وزن های اولیه را نیز به صورت رندوم به صورت زیر تایین کردیم:



سپس برای epoch اول، برای هر نورون به ترتیب مقدار خروجی را بدست میاوریم که مقادیر در شکل زیر آورده شده و در آخر loss را با استفاده از mse بدست آورده ایم که برابر با 0.1681 شد.



حال با استفاده از الگوریتم بک backpropagation مقدار خطا را برای هر کدام از نورون ها به صورت زنجیره ای بدست میاوریم.

در ابتدا مشتق تابع loss را نسبت به خروجی تابع sigmoid بدست میاوریم. با توجه به اینکه در این مسئله فقط یک دیتا ورودی داریم، تابع لاس به صورت رو به روست:

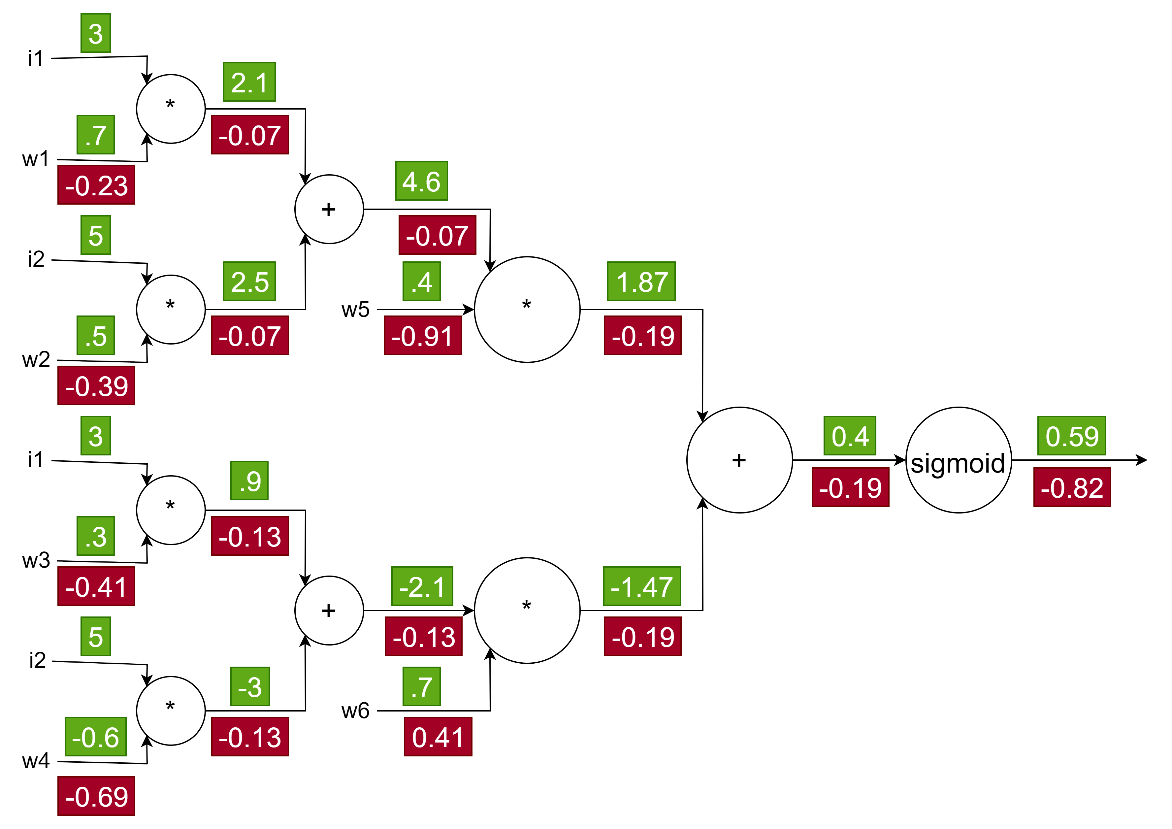
که مشتق آن نصبت به y هت به صورت زیر بدست میاید:

*در قدم بعدی باید مشتق تابع سیگموید را نسبت به* z *بگیریم که در تمرین 2 بدست آورده بودیم، به صورت رو به رو بدست میاید:*

*حال اگر بخواهید مشتق لاس را نسبت به z بدست بیاوریم، طبق قاعه زنجیره ای باید مشتق لاس نسبت به سیگموید را ضرب در مشتق سیگموید به z کنیم که برابر با* -0.198358 *میشود که به طور تقریبی برابر با -0.19 در نظر میگیریم. بعد از این دو نورون، کار آسون میشود، برای نورون های جمع، جواب مشتق برابر همان عدد قبل میشود و برای ضرب، نورون های ورودی به صورت برعکس ضرب میشوند. مثال:*

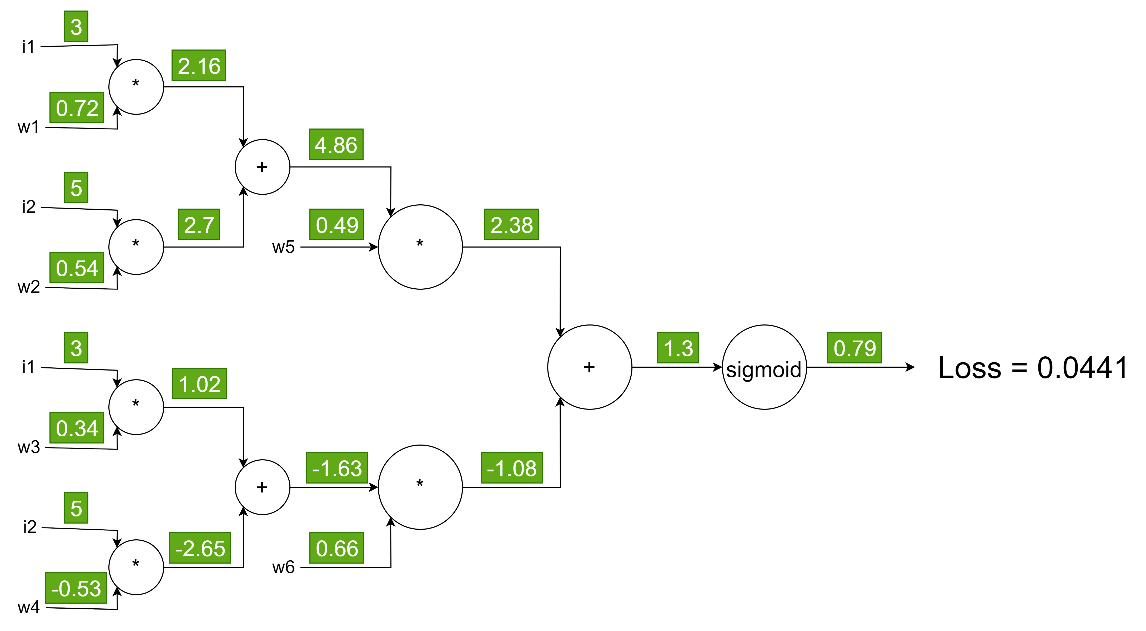
**

*به همین ترتیب خطای مربوط به هرکدام از نورون ها را بدست آوردیم که در شکل زیر مشاهده میکنید:*

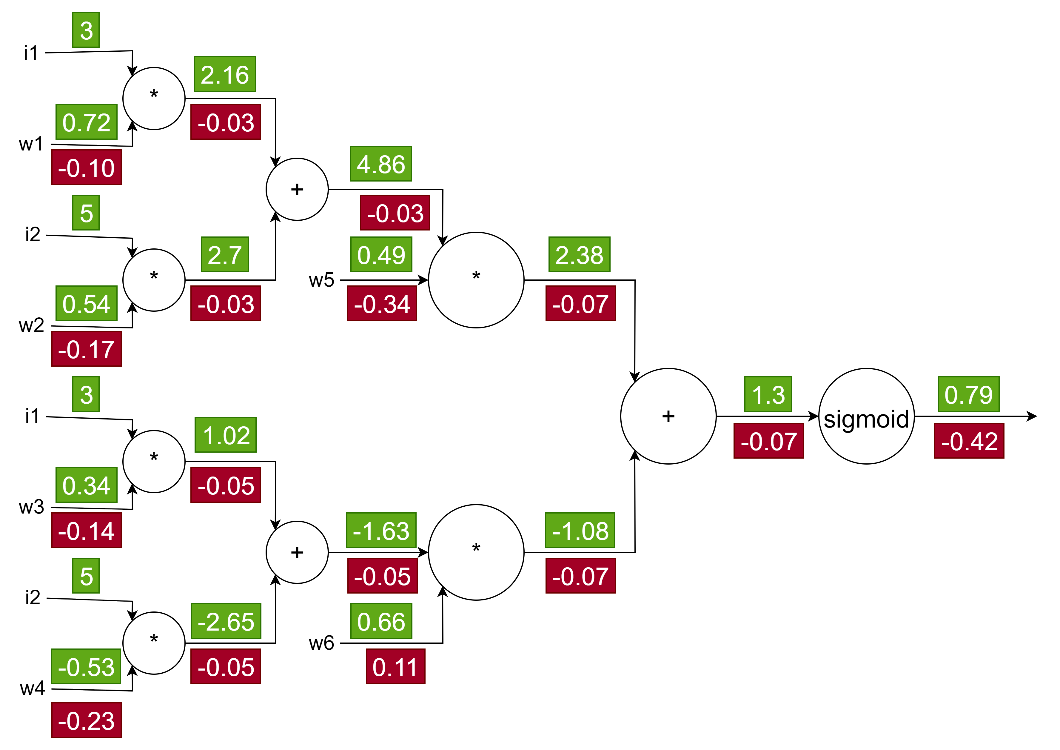
**

*حال میخواهیم وزن هارا با توجه به مقدار خطا و با استفاده از روش گرادیان آپدیت کنیم. نرخ یادگیری را برابر با 0.1 در نظر میگیرم. وزن ها به صورت زیر آپدیت میشوند:*

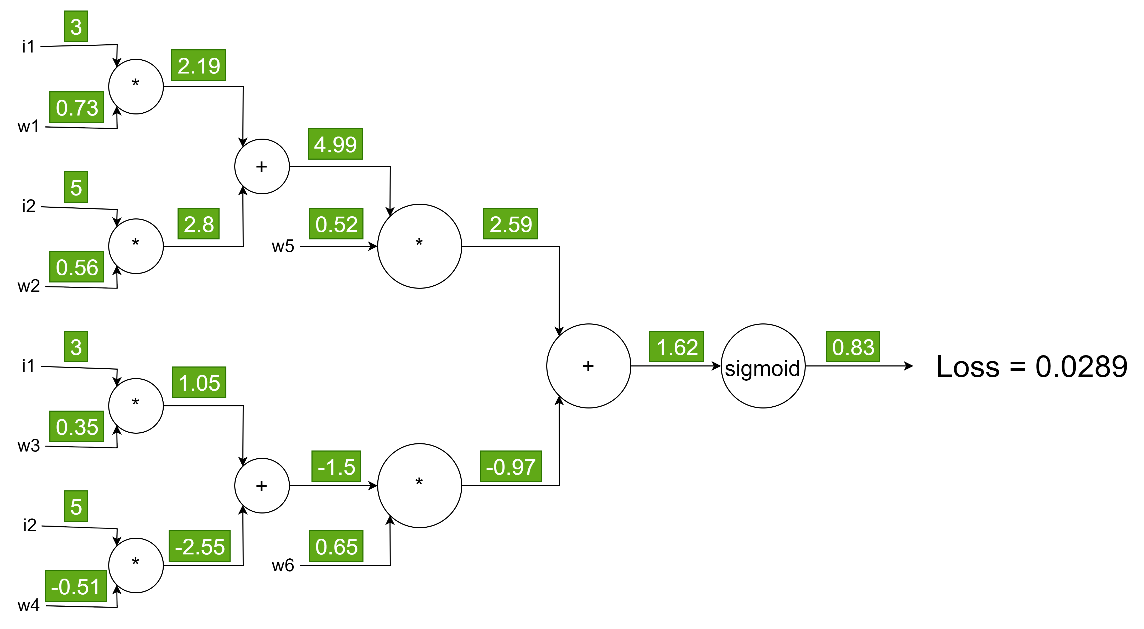
*بقیه w ها را نیز به صورت بالا آپدیت میکنیم. در ادامه نیز وارد epoch 2 شده و دوباره مانند قبل خروجی هر نورون را بدست میاوریم. Epoch 2 با استفاده از w های آپدیت شده در شکل زیر قابل مشاهده است:*

**

*حال باید برای epoch 2 الگوریتم backpropagation را اجرا کنیم که محاسبه مانند مرحله ی قبل است و خروجی را در زیر میبینیم:*

**

*وزن ها را آپدیت کرده و برای آخرین بار لاس را حساب میکنیم:*

**

*همانطور که مشاهده شد، لاس در epoch اول برابر با 0.16 ، در epoch دو برابر با 0.04 و در epoch سوم برابر با 0.02 شد که نشان میدهد الگوریتم به درستی دارد بهینه میکند و لاس در هر epoch پایین میاید. درضمن مشاهده میکنیم که فاصله ی لاس در epoch اول و دوم نسبت به epoch دوم به سوم بیشتر است که میتوان نتیجه گرفت که شیب کم شده و الگوریتم به سمت همگرایی میرود.*

1. *در ابتدا و بعد از import ها تنظیمات مورد نیاز برای مدل را مقدار دهی کرده ایم و همچنین چک میشود که آیا gpu در دسترس است یا خیر.*

*در ادامه دیتا ها را از ورودی گرفته و ترنسفورم هایی مانند نرمالایز کردن را روی آن ها اجرا میکنیم. برای ترنسفورم دیتای train ترنسفورم های RandomCrop، RandomHorizontalFlip را نیز علاوه بر ترنسفورم های مربوط به نرمالایز کردن و تنسور کردن را اضافه کردیم که به تریتب، قسمت هایی از تصاویر را برش میدهد تا پدینگ 4 و دومی نیز به صورت افقی و رندوم تصویر را میچرخواند که باعث این ترنسفورم ها باعث اضافه شدن دیتای ورودی شده و از با بالا رفتن نویز تصاویر، احتمال اور فیت شدن مدل کمتر میشود. و در ادامه نیز تعدادی از داده های train را به همراه لیبلشان نمایش داده ایم.*

*برای ساخت مدل، در لایه های conv2d ابعاد چنل را دو برابر میکنیم ولی ابعدا تصویر ثابت میماند. در لایه های MaxPool2d چنل وردی ثابت نگه میداریم و ابعاد را نصف میکنیم. (stride = 2) در آخر (fc\_layer) نیز سه لایه با ابعاد 512، 256 و 10 برای دسته بندی قرار میدهیم. برای همه ی لایه های dropout احتمال را برابر با 0.05 در نظر گرفتیم.*

*برای تابع ضرر و optimizer نیز همانطور که گفته شد به ترتیب از croos entropy و adam استفاده شده است که لرنینگ ریت را برابر با 0.001 قرار دادیم.*

*حال به ازای هر epoch روی trainloader که داده ها را به صورت batch های 128 تایی (در ابتدا مشخص کردیم) به ما میدهد، فور میزنیم. در صورت استفاده از gpu داده ورودی را برای استفاده از آن در نظر میگیریم. سپس دیتای train را به مدل داده و به صورت forward حرکت میکنیم، بعد لاس را محاسبه کرده و در آخر مقادیر adam را صفر میکنیم و با مقدار جدید loss وزن ها را آپدیت میکنیم. در انتهای فور به اضای یک مقدار خاص از ایتریشن ها، مقدار لاس را خروجی میدهیم.*

*در سل آخر نیز به ازای هر لیبل دقت آن را محاسبه میکنیم که با توجه به کم بودن epoch ها نتایج خوبی دست نیامد.*

*منابع:*

[*https://androidkt.com/convolutional-neural-network-using-sequential-model-in-pytorch/*](https://androidkt.com/convolutional-neural-network-using-sequential-model-in-pytorch/)

[*https://pytorch.org/tutorials/beginner/blitz/cifar10\_tutorial.html*](https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html)

[*https://zhenye-na.github.io/2018/09/28/pytorch-cnn-cifar10.html#loss-training-accuracy-and-test-accuracy*](https://zhenye-na.github.io/2018/09/28/pytorch-cnn-cifar10.html#loss-training-accuracy-and-test-accuracy)

[*https://pytorch.org/vision/stable/transforms.html*](https://pytorch.org/vision/stable/transforms.html)

[*https://modelzoo.co/model/data-augmentation-and-sampling-for-pytorch*](https://modelzoo.co/model/data-augmentation-and-sampling-for-pytorch)