به نام خدا

درس یادگیری عمیق

تمرین اول

مدرس : دکتر داوودآبادی

دستیاران : سحرسرکار، فائزه صادقی، حسن حماد

دانشجو : سارا سادات یونسی / ۹۸۵۳۳۰۵۳

فهرست

سوال ۱

الف) بیش برازش مفهومی در علم داده است که زمانی اتفاق می افتد که یک مدل آماری دقیقاً با داده های آموزشی آن مطابقت داشته باشد .وقتی این اتفاق می‌افتد، متأسفانه الگوریتم نمی‌تواند به طور دقیق در برابر داده‌های دیده نشده عمل کند و هدف خود را شکست دهد.

مشکل overfitting در شبکههای عصبی این است که مدل به جای یادگیری الگوی کلی دادهها، سعی میکند به دقت با دادههای آموزشی مطابقت پیدا کند. این باعث میشود که مدل نتواند به خوبی با دادههای جدید و دیده نشده که ممکن است از دادههای آموزشی متفاوت باشند، سازگار شود. بنابراین، مدل از توانایی تعمیم خود از دست میدهد و دچار کاهش عملکرد میشود. مشکل overfitting میتواند به دلایل مختلفی ایجاد شود، از جمله:

• پیچیدگی بیش از حد مدل: اگر مدل دارای تعداد زیادی پارامتر یا ویژگی باشد، ممکن است بتواند با دادههای آموزشی بیش از حد fit شود و نویزها یا جزئیات غیرمرتبط را نیز یاد بگیرد. برای حل این مشکل، میتوان تعداد پارامترها یا ویژگیهای مدل را کاهش داد یا از روشهای منظمسازی (regularization) استفاده کرد. منظمسازی یک روش است که با افزودن یک جمله جریمه به تابع هزینه، سعی میکند پیچیدگی مدل را کنترل کند و وزنهای بزرگ را جلوگیری کند. برای مطالعه بیشتر درباره منظمسازی میتوانید به منبع مراجعه کنید.

• کمبود دادههای آموزشی: اگر مجموعه دادههای آموزشی کوچک یا نامتعادل باشد، ممکن است مدل نتواند الگوی کلی دادهها را یاد بگیرد و فقط با دادههای موجود سازگار شود. برای حل این مشکل، میتوان تعداد دادههای آموزشی را افزایش داد یا از روشهای افزایش داده (data augmentation) استفاده کرد. افزایش داده یک روش است که با اعمال تغییرات کوچک یا تصادفی بر روی دادههای موجود، مجموعه دادههای جدید و متنوع تولید میکند.

مشکل underfitting در شبکه های عصبی این است که مدل نتواند الگوی کلی دادهها را یاد بگیرد و فقط با دادههای ساده و خطی سازگار شود. این باعث میشود که مدل از توانایی تعمیم خود از دست بدهد و دچار کاهش عملکرد میشود. مشکل underfitting میتواند به دلایل مختلفی ایجاد شود، از جمله:

• سادگی بیش از حد مدل: اگر مدل دارای تعداد کمی پارامتر یا ویژگی باشد، ممکن است نتواند با دادههای پیچیده و غیرخطی مطابقت پیدا کند و نویزها یا جزئیات مرتبط را نادیده بگیرد. برای حل این مشکل، میتوان تعداد پارامترها یا ویژگیهای مدل را افزایش داد یا از روشهای انتخاب ویژگی (feature selection) یا استخراج ویژگی (feature extraction) استفاده کرد. انتخاب ویژگی یک روش است که با حذف ویژگیهای بیاهمیت یا تکراری، سعی میکند مجموعه ویژگیهای بهینه را برای مدل انتخاب کند. استخراج ویژگی یک روش است که با ترکیب ویژگیهای موجود، سعی میکند مجموعه ویژگیهای جدید و معنادار را برای مدل ایجاد کند.

دلایل ایجاد بیش‌برازش(Overfitting)

بیش‌برازش (Overfitting) می‌تواند به یکی از این دلایل اتفاق بیفتد:

- مدل بیش‌ازحد پیچیده است و ویژگی‌های هم‌خط (Collinear) را دربرمی‌گیرد که واریانس داده‌های ما را افزایش می‌دهد؛
- تعداد ویژگی‌های داده‌های ما بیشتر یا برابر با تعداد داده است؛
- حجم داده بسیار کم است.
- داده پیش‌پردازش نشده تمیز نیست و نویز (Noise) دارد.

راه‌حل‌های مقابله با بیش‌برازش

بیش برازش (Overfitting) مسئله‌ای بسیار رایج در یادگیری ماشین است .روش‌های مختلفی هم برای جلوگیری از آن وجود دارد. در این بخش به این روش‌ها اشاره خواهیم کرد.

مدلی که به مشکل کم‌برازش دچار است به نتایج اشتباه در داده‌های جدید را که روی آن‌ها آموزش داده نشده است رقم خواهد زد و اغلب حتی درمورد داده‌های آموزشی نیز عملکرد ضعیفی دارد. زمانی که مدل دچار مشکل کم برازش (Underfitting)است بسیاری از ویژگی‌های داده‌های آموزشی را را نادیده می‌گیرد و نمی‌تواند رابطه‌ی میان ورودی و خروجی را یاد بگیرد.

بیش برازش (Overfitting) یکی از خطاهای مدل‌سازی در علم داده (Data Science) است. این خطا هنگامی اتفاق می‌افتد که مدل ویژگی‌های داده‌های آموزشی را به‌جای یادگیری، حفظ کرده باشد، یعنی بیش‌ازحد روی آن آموزش دیده باشد؛ درنتیجه، این مدل فقط در مجموعه‌ی داده‌های آموزشی مفید خواهد بود و نه در مجموعه‌ی داده‌های دیگر که هنوز آن‌ها را ندیده است.

ب) بهترین روش برای تشخیص مدل‌های اضافه برازش، آزمایش مدل‌های یادگیری ماشین بر روی داده‌های بیشتر با نمایش جامع مقادیر و انواع داده‌های ورودی ممکن است .به طور معمول، بخشی از داده های آموزشی به عنوان داده های تست برای بررسی بیش از حد مناسب استفاده می شود .ضریب خطای بالا در داده های تست نشان دهنده بیش از حد برازش است.

اگر خطای ما روی داده های اموزشی کم باشد و مدل عملکرد خوبی داشته باشد ولی روی داده های تست به خوبی عمل نکند می فهمیم که فقط روی داده های اموزشی اورفیت شده است و در تست و پیش بینی به خوبی عمل نمی کند.

اگر در نمودار های ارزیابی ما دقت و لاس بررسی کنیم می توانیم به اورفیت پی ببریم

اگر اورفیت شود دقت در داده های اموزش زیاد است و خطا کم اما در تست برعکس است.

یا از داده های اعتبار سنجی می توان استفاده کرد اگر با اموزش دادن مدل و بررسی کردن اینکه دقت در داده های اموزش افزایش یافته باید در اعتبار سنجی هم افزایش یابد تا نماد اورفیت نباشد

پ)توضیحی برای dropout در یادگیری ماشینی، "drop out" به تمرین نادیده گرفتن گره های خاص در یک لایه به طور تصادفی در طول آموزش اشاره دارد. Drop out یک رویکرد منظم سازی است که با اطمینان از اینکه هیچ واحدی به یکدیگر وابسته نیست، از تطبیق بیش از حد جلوگیری می کند.

در مرحله آموزش :

در آن نورون‌های انتخابی تصادفی در طول تمرین نادیده گرفته می‌شوند. آنها به طور تصادفی "انصراف داده می شوند". این به این معنی است که سهم آنها در فعال سازی نورون های پایین دست به طور موقت در گذر رو به جلو حذف می شود و هیچ گونه به روز رسانی وزنی برای نورون در عبور به عقب اعمال نمی شود.

حال با اعمال ماسک دراپ اوت در مرحله ی اموزش فقط نورون هایی که ماسک ۱ داشتند در فرایند اموزش تاثیر داده می شوند

و ماسک های صفر غیر فعال می شوند

۱,۹	۰	۰	۱,۶
۰	۲,۵	۲,۵	۰
۰	۳,۲	۳,۷	۰
۱,۲	۰	۰	۱,۳

در مرحله تست :

در زمان تست، تمام نورون ها فعال هستند اما خروجی هر نورون را در ضریب p ضرب می کنیم

در اینجا تعداد ۰ = تعداد ۱ = ۰,۵

۰,۹۵	۰,۱-	۰,۳۵-	۰,۸
۰,۴۵-	۱,۲۵	۱,۲۵-	۱,۱۵-
۰,۲-	۱,۸۵	۱,۶	۰,۲۵-
۰,۶	۱,۳-	۰,۲-	۰,۶۵

با جمع کردن هر قسمت می توان به مقدار توتال ان قسمت رسید

سوال ۲)

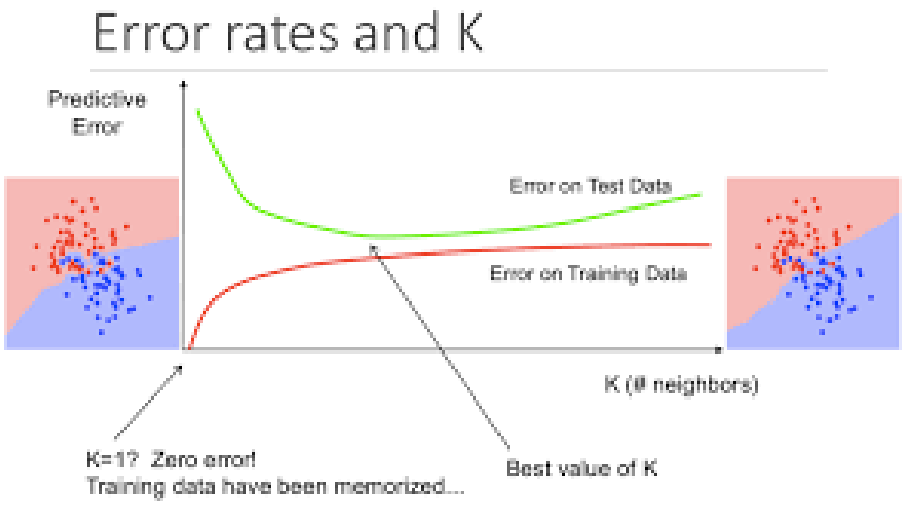
الف) اگر مقادیر مختلف k را در نظر بگیریم، می‌توانیم مبادله بین بایاس و واریانس را مشاهده کنیم .با افزایشk ، مدل پایدارتری داریم، یعنی واریانس کوچکتر، با این حال، بایاس نیز افزایش می یابد .با کاهشk ، بایاس نیز کاهش می یابد، اما مدل کمتر پایدار است.

با تغییر مقدار K در الگوریتم نزدیک ترین همسایگی، بایاس و واریانس مدل نیز تغییر می کنند. بایاس به اختلاف بین مقدار مورد انتظار پیش بینی مدل و مقدار واقعی هدف اشاره دارد. واریانس به تغییرات پیش بینی مدل در برابر تغییرات داده های آموزشی اشاره دارد. به طور کلی، هنگامی که مقدار K کاهش می یابد، بایاس مدل کاهش می یابد و واریانس مدل افزایش می یابد. به عبارت دیگر، مدل حساس تر به نویز و تغییرات داده های آموزشی می شود. هنگامی که مقدار K افزایش می یابد، بایاس مدل افزایش می یابد و واریانس مدل کاهش می یابد. به عبارت دیگر، مدل کمتر به داده های آموزشی وابسته می شود ولی ممکن است الگوی کلی داده ها را نادیده بگیرد. بنابراین، انتخاب یک مقدار مناسب برای K می تواند توازنی بین بایاس و واریانس مدل ایجاد کند.

ا افزایش K، KNN منحنی صاف تری را با داده ها مطابقت می دهد. این به این دلیل است که مقدار بالاتر K با در نظر گرفتن داده‌های بیشتر، لبه را کاهش می‌دهد و در نتیجه پیچیدگی و انعطاف‌پذیری کلی مدل را کاهش می‌دهد.

مقادیر کمتر k می تواند واریانس بالایی داشته باشد، اما بایاس کم و مقادیر بزرگتر k ممکن است منجر به بایاس زیاد و واریانس کمتر شود. انتخاب k تا حد زیادی به داده‌های ورودی بستگی دارد، زیرا داده‌هایی با مقادیر پرت یا نویز بیشتر احتمالاً با مقادیر بالاتر k عملکرد بهتری خواهند داشت.

ر جایی که k بزرگتر است، فاصله بزرگتر می شود، که این اصل پشت kNN را شکست می دهد - اینکه همسایه هایی که نزدیکتر هستند چگالی یا کلاس های مشابهی دارند. به طور معمول یک k بهینه وجود دارد که می توانید آن را با استفاده از اعتبارسنجی متقاطع پیدا کنید - نه خیلی بزرگ و نه خیلی کوچک



https://medium.com/30-days-of-machine-learning/day-3-k-nearest-neighbors-and-bias-variance-tradeoff-75f84d515bdb

How does K affect bias and variance in KNN?
Lower values of k can have high variance, but low bias, and larger values of k may lead to high bias and lower variance. The choice of k will largely depend on the input data as data with more outliers or noise will likely perform better with higher values of k.

ب)

درست

بله، استفاده از منظمسازی ممکن است در برخی موارد باعث تضعیف عملکرد مدل شود. منظمسازی یک روش است که با افزودن یک جمله جریمه به تابع هزینه، سعی میکند پیچیدگی مدل را کنترل کند و وزنهای بزرگ را جلوگیری کند. این کار میتواند باعث کاهش خطای آموزش و افزایش توانایی تعمیم مدل شود. اما اگر منظمسازی بیش از حد انجام شود، ممکن است باعث شود که مدل نتواند الگوی کلی دادهها را یاد بگیرد و دچار بایاس زیاد شود. بنابراین، انتخاب یک مقدار مناسب برای ضریب منظمسازی اهمیت زیادی دارد.

منظم سازی پاسخی به بیش از حد مناسب است .این تکنیکی است که دقت مدل را بهبود می بخشد و همچنین از از دست رفتن داده های مهم به دلیل عدم تناسب جلوگیری می کند

منظم‌سازی عملکرد مجموعه داده‌ای را که الگوریتم برای یادگیری پارامترهای مدل (وزن ویژگی) استفاده می‌کند، بهبود نمی‌بخشد .با این حال، می تواند عملکرد تعمیم را بهبود بخشد، به عنوان مثال، عملکرد در داده های جدید و دیده نشده، که دقیقا همان چیزی است که ما می خواهیم

.

غلط

افزودن بسیاری از ویژگی‌های جدید به مدل over روی مجموعه آموزشی کمک می‌کند .افزودن بسیاری از ویژگی‌های جدید مدل‌های گویاتری را به ما می‌دهد که می‌توانند بهتر با مجموعه آموزشی ما مطابقت داشته باشند .اگر ویژگی های جدید بیش از حد اضافه شود، باعث افزایش مدل می شود و مدل حفظ می کند و نمی تواند تعمیم بدهد این می تواند منجر به تطبیق بیش از حد مجموعه آموزشی شود.و باعث اورفیتینگ شود

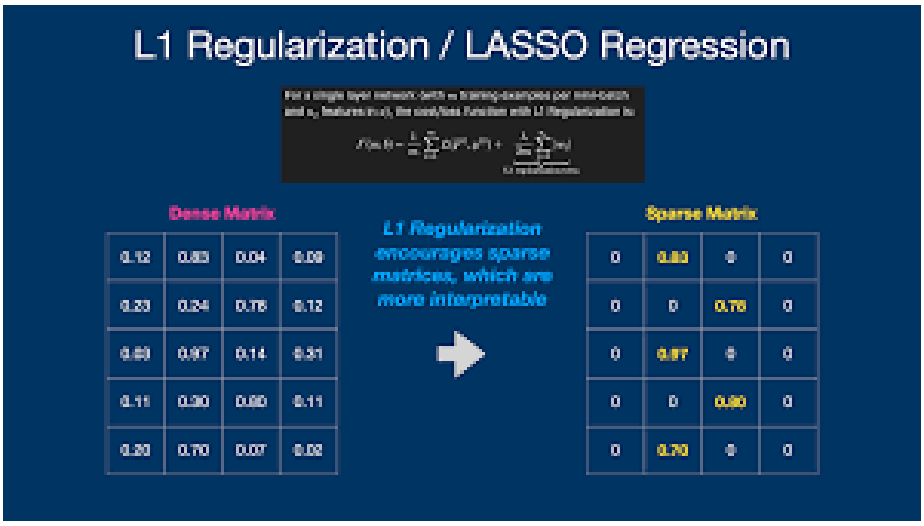
غلط

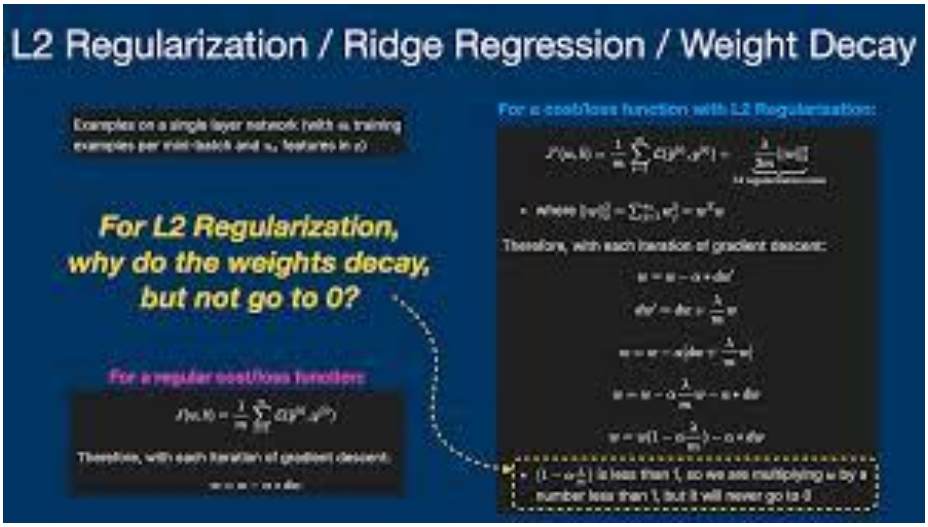
هنگامی که ضریب تنظیم کمتر باشد، پارامترها به ۰ نزدیکتر می شوند و در نتیجه برازش کم می شود .اگر ضریب تنظیم کوچکتر باشد، تقریباً هیچ جریمه ای برای پارامتر وجود ندارد که منجر به بیش از حد برازش می شود

به طور خلاصه، منظم‌سازی در یادگیری ماشینی، فرآیند منظم‌سازی پارامترهایی است که تخمین‌های ضریب را محدود، منظم یا به سمت صفر کاهش می‌دهد. به عبارت دیگر، این تکنیک از یادگیری یک مدل پیچیده تر یا انعطاف پذیرتر جلوگیری می کند و از خطر بیش از حد برازش جلوگیری می کند.

پ)

Comparison of L1 and L2 regularization	
L1 regularization	***L2 regularization***
Sum of absolute value of weights	Sum of square of weights
Sparse solution	Non-sparse solution
Multiple solutions	One solution
Built-in feature selection	No feature selection
Robust to outliers	Not robust to outliers (due to the square term)





توضیحی برای l2:

وزن‌های بزرگ در یک شبکه عصبی اغلب نشانه‌ای از یک شبکه بیش از حد پیچیده است که داده‌های آموزشی بیش از حد مناسب است. بنابراین، یکی از راه های جلوگیری از پیچیده شدن بیش از حد یک مدل، جلوگیری از بزرگ شدن وزنه ها است.

منظم‌سازی L2 تکنیکی برای کاهش پیچیدگی مدل با کاهش وزن‌های یک مدل متناسب با مجذور هر وزن است (بنابراین به‌ویژه آنهایی که بالاترین را دارند جریمه می‌شوند)، اما آنها را ۰ نمی‌کند.

L2 Regularization قدر مجذور وزن را به عنوان جریمه به تابع ضرر اضافه می کند (ضرب در یک فراپارامتر لامبدا). به دلیل مشتق، این بدان معناست که در طول شیب نزول، وزن‌هایی که بالاتر هستند بیشتر جریمه می‌شوند، در حالی که وزن‌های پایین‌تر آنقدر تغییر نمی‌کنند (بنابراین مانع از رفتن آنها به صفر می‌شود).

توضیحی برای L1:

برای هر مدل معینی، برخی از وزنه ها از بقیه مهمتر خواهند بود. با این حال، سر و صدای تصادفی در حین تمرین باعث می شود که برخی از وزنه های کم اهمیت تأثیر بگذارند. بنابراین، یکی از راه‌های جلوگیری از تطبیق بیش از حد مدل به سر و صدا (و همچنین آسان‌تر کردن تشخیص اینکه کدام ویژگی‌ها واقعاً مهم هستند) خلاص شدن از شر وزنه‌هایی است که کمتر غالب هستند.

منظم‌سازی L1 تکنیکی است برای کاهش پیچیدگی مدل با به صفر رساندن وزن‌های کم‌اهمیت (یعنی پراکندگی را تشویق می‌کند)، در نتیجه یک مدل را به صورت بصری قابل تفسیرتر می‌کند.

تنظیم L1 مقدار مطلق یک وزن را به عنوان جریمه به تابع ضرر اضافه می کند (ضرب در یک فراپارامتر لامبدا). این بدان معناست که در طول شیب نزول، همه وزنه‌ها به طور مکرر جریمه می‌شوند، به طوری که فقط وزنه‌هایی که مهم هستند (یعنی به طور مکرر در جهت مثبت افزایش می‌یابند، زیرا نمونه‌های تمرینی از آنها استفاده می‌کنند، نه فقط به دلیل نویز تصادفی) زنده می‌مانند، و بقیه وزن‌ها باقی خواهند ماند. برو به ۰

با توجه به توضیحات بالا نمونه هایی که داخل خود برخی صفر دارند از l1 پیروی و بقیه از که کوچک شده ولی به صفر نرسیده L2 پیروی میکنند

L2

L1

L2

L1

سوال ۳)

الف)ت قطیر دانش بیشتر بر روی مدل‌های شبکه عصبی مرتبط با معماری‌های پیچیده از جمله چندین لایه و پارامترهای مدل انجام می‌شود .بنابراین، با ظهور یادگیری عمیق در دهه اخیر و موفقیت آن در زمینه های مختلف از جمله تشخیص گفتار، تشخیص تصویر و پردازش زبان طبیعی، تکنیک های تقطیر دانش برای کاربردهای عملی در دنیای واقعی اهمیت پیدا کرده است . چالش استقرار مدل‌های شبکه عصبی عمیق به‌ویژه برای دستگاه‌های لبه‌ای با حافظه محدود و ظرفیت محاسباتی مرتبط است . برای مقابله با این چالش، ابتدا یک روش فشرده سازی مدل برای انتقال دانش از یک مدل بزرگ به آموزش یک مدل کوچکتر بدون کاهش قابل توجهی در عملکرد، پیشنهاد شد.

تقطیر دانش در یادگیری عمیق یک روش است که با استفاده از یک مدل بزرگ و پیچیده به عنوان معلم، یک مدل کوچک و ساده را به عنوان دانشآموز آموزش میدهد. هدف این روش انتقال دانش و توانایی تعمیم پذیری از مدل معلم به مدل دانشآموز است. این کار میتواند باعث کاهش حجم، زمان و منابع مورد نیاز برای اجرای مدل دانشآموز شود، بدون اینکه دقت آن به شدت کاهش یابد. برای این منظور، مدل دانشآموز از توزیع احتمال خروجی مدل معلم به عنوان اهداف نرم و از برچسب های داده های آموزشی به عنوان اهداف سخت استفاده میکند.

- کاهش حجم، زمان و منابع مورد نیاز برای اجرای مدل دانشآموز روی دستگاههای محدود مانند موبایل.
- افزایش دقت و توانایی تعمیمپذیری مدل دانشآموز با استفاده از دانش و ویژگیهای مدل معلم
- کاهش بیشبرازش مدل دانشآموز با استفاده از اهداف نرم که توزیع احتمال خروجی مدل معلم را نشان میدهند
- امکان تقسیم وظایف بین چندین مدل دانشآموز که هر کدام به یک دسته از دادهها تخصص دارند

Knowledge distillation is a method of transferring the knowledge from a complex deep neural network (DNN) to a smaller and faster DNN, while preserving its accuracy. Recent variants of knowledge distillation include teaching assistant distillation, curriculum distillation, mask distillation, and decoupling distillation, which aim to improve the performance of knowledge distillation by introducing additional components or by changing the learning process. Teaching assistant distillation involves an intermediate model called the teaching assis- tant, while curriculum distillation follows a curriculum similar to human education. Mask distillation

focuses on transferring the attention mechanism learned by the teacher, and decoupling distillation decouples the distillation loss from the task loss. Overall, these variants of knowledge distillation have shown promising results in improving the performance of knowledge distillation.

ب) با استفاده از احتمالات تولید شده برای هر کلاس که با عنوان "اهداف نرم" توسط مدل سنگین و به منظور آموزش مدل کوچک تولید می‌شوند می‌توان توانایی تعمیم‌پذیری مدل سنگین را به یک مدل کوچک‌تر انتقال داد. در مرحله‌ی انتقال می‌توانیم را برای آموزش مدل سنگین به کار ببریم. وقتی مدل سنگین مجموعه‌ای از "مجموعه انتقال" همان مجموعه‌ی آموزشی یا یک مدل‌های ساده‌تر باشد می‌توانیم از میانگین حسابی یا هندسی توزیع‌های پیش‌بین هر یک از آن مدل‌ها به عنوان اهداف نرم استفاده کنیم. زمانی که آنتروپی اهداف نرم بالا باشد، برای آموزش هر مورد اطلاعات بسیار بیشتر و واریانس خیلی کمتری بین آن‌ها ارائه می‌دهند (در مقایسه با اهداف سخت). بنابراین مدل کوچک می‌تواند روی داده‌های بسیار کمتری از آنچه مدل سنگین نیاز دارد، آموزش ببیند و در عین حال به نرخ یادگیری بالاتری دست یابد

ر تقطیر، دانش از مدل معلم به دانش آموز با به حداقل رساندن یک تابع ضرر که در آن هدف، توزیع احتمالات کلاس پیش بینی شده توسط مدل معلم است، منتقل می شود. یعنی - خروجی یک تابع softmax روی لاجیت های مدل معلم. با این حال، در بسیاری از موارد، این توزیع احتمال دارای کلاس صحیح با احتمال بسیار بالا است، با تمام احتمالات کلاس های دیگر بسیار نزدیک به ۰. به این ترتیب، اطلاعات زیادی فراتر از برچسب های حقیقت زمینی ارائه شده در مجموعه داده ارائه نمی دهد. برای مقابله با این موضوع، هینتون و همکاران، ۲۰۱۵ مفهوم "دمای softmax" را معرفی کردند. احتمال پی

از کلاس I از logits z محاسبه می شود.

$$p_i = \frac{exp\left(\frac{z_i}{T}\right)}{\sum_j \exp\left(\frac{z_j}{T}\right)}$$

جایی که تی

پارامتر دما است. زمانی که T=1

تابع softmax استاندارد را دریافت می کنیم. همانطور که تی

رشد می کند، توزیع احتمال تولید شده توسط تابع softmax نرم تر می شود و اطلاعات بیشتری در مورد اینکه معلم کدام کلاس ها را شبیه به کلاس پیش بینی شده پیدا کرده است، ارائه می دهد. هینتون این را "دانش تاریک" تعبیه شده در مدل معلم می نامد، و این دانش تاریک است که ما در فرآیند تقطیر به مدل دانش آموز منتقل می کنیم. هنگام محاسبه تابع ضرر در مقابل اهداف نرم معلم، از همان مقدار T استفاده می کنیم

برای محاسبه softmax روی logits دانش آموز. ما این تلفات را "از دست دادن تقطیر" می نامیم.

هینتون و همکاران، ۲۰۱۵ دریافتند که آموزش مدل تقطیر شده برای تولید برچسب‌های صحیح (بر اساس حقیقت زمین) علاوه بر برچسب‌های نرم معلم نیز مفید است. از این رو، ما تلفات «استاندارد» بین احتمالات کلاس پیش‌بینی‌شده دانش‌آموز و برچسب‌های حقیقت پایه (که «برچسب‌ها/هدف‌های سخت» نیز نامیده می‌شود) را محاسبه می‌کنیم. ما این فقدان را "از دست دادن دانش آموز" می نامیم. هنگام محاسبه احتمالات کلاس برای از دست دادن دانش آموز از T=1 استفاده می کنیم

.

تابع تلفات کلی، که هم تلفات تقطیر و هم تلفات دانشجویی را در بر می گیرد، به صورت زیر محاسبه می شود:

$$\mathcal{L}(x;W) = \alpha * \mathcal{H}(y, \sigma(z_s; T=1)) + \beta * \mathcal{H}(\sigma(z_t; T=\tau), \sigma(z_s, T=\tau))$$

where x is the input, W are the student model parameters, y is the ground truth label, $\mathcal{H}$ is the cross-entropy loss function, σ is the softmax function parameterized by the temperature T, and α and β are coefficients. z_s and z_t are the logits of the student and teacher respectively.

پ) تلفات نهایی با میانگین گیری دو تلفات با وزن قابل توجه (آلفا) داده شده به تلفات تقطیر برای دستیابی به بهینه محاسبه می شود. اولین تابع تلفات تقطیر از اهداف نرم شبکه معلم برای بهینه سازی برچسب های خروجی شبکه دانش آموز استفاده می کند. دومین تلفات مبتنی بر آنتروپی متقابل (یا تابع از دست دادن دانش‌آموز) از برچسب‌های حقیقت پایه استفاده می‌کند و آنها را با پیش‌بینی‌های دانش‌آموز مقایسه می‌کند.

Distillation loss uses the soft targets to minimize the squared difference between the logits produced by the cumbersome model and the logits produced by the small model

https://intellabs.github.io/distiller/knowledge_distillation.html

https://intellabs.github.io/distiller/knowledge_distillation.html

.

سوال ۴)

توضیح کد :

ین توابع بخشی از کد یک مدل شبکه عصبی هستند که با استفاده از پایتورچ نوشته شده‌اند. این توابع هدفشان این است که را برای مدل مقداردهی اولیه کنند. برای این (RMS) و مقادیر میانگین مربعات (momentum) پارامترها، مقادیر لحظهای :استفاده میشوند. توضیح هر تابع به شرح زیر است numpy منظور، از توابع تولید عدد تصادفی از کتابخانه

self._initalize_parameters(): ·این تابع وزنها و بایاسهای مدل را با مقادیر تصادفی از توزیع نرمال مقداردهی اولیه میکند. این تابع از تابع np.random.randn استفاده میکند که یک تنسور با ابعاد دلخواه از توزیع نرمال استاندارد تولید این تابع از این توزیع برای تولید وزنها و بایاسها استفاده میکند و آنها را در دیکشنری parameters ذخیره میکند.

self._initalize_moms(): ·این تابع مقادیر لحظهای (momentum) را برای مدل مقداردهی اولیه میکند. مقادیر لحظهای نشاندهندهی گشتاور یا سرعت تغییر وزنها در الگوریتم بهینهسازی هستند. این تابع از تابع np.zeros استفاده میکند که یک تنسور با ابعاد دلخواه از صفر تولید میکند .initialise-1-layer-neural-network-parameters- این تابع از این تنسور برای مقداردهی اولیهی مقادیر لحظهای استفاده میکند و آنها را در دیکشنری moms ذخیره میکند.

self._initalize_RMSs(): ·این تابع مقادیر میانگین مربعات (RMS) را برای مدل مقداردهی اولیه میکند. مقادیر میانگین مربعات نشاندهندهی میزان تغییرات وزنها در الگوریتم بهینهسازی هستند. این تابع از تابع np.zeros استفاده میکند که یک تنسور با ابعاد دلخواه از صفر تولید .initialise-1-layer-neural-network-parameters این تابع از این تنسور برای مقداردهی اولیهی مقادیر میانگین مربعات استفاده میکند و آنها را در دیکشنری RMSs ذخیره میکند.

- def random_tensor(self, size): این بخش تعریف تابع random_tensor را نشان میدهد که یک پارامتر به نام size دریافت میکند. self نشاندهندهی این است که این تابع متعلق به یک کلاس است که مدل شبکه عصبی را نمایندگی میکند.

- return (torch.randn(size)): این بخش خروجی تابع را نشان میدهد که یک تنسور با ابعاد size است.
torch.randn(size) یک تابع از کتابخانه پایتورچ است که یک تنسور با ابعاد دلخواه از توزیع نرمال استاندارد تولید میکند.

- .requires_grad(): این بخش یک ویژگی از تنسورهای پایتورچ است که مشخص میکند که آیا گرادیان این تنسور در هنگام بهینهسازی محاسبه و ذخیره شود یا خیر. اگر این ویژگی برابر True باشد، گرادیان این تنسور محاسبه و ذخیره میشود و اگر برابر False باشد، گرادیان این تنسور محاسبه و ذخیره نمیشود. در اینجا، این ویژگی برابر True قرار داده شده است، زیرا میخواهیم گرادیان این تنسور را برای بهروزرسانی وزنها و بایاسهای مدل استفاده کنیم.

• def _initalize_moms(self): این تابع مقادیر لحظهای (momentum) را برای مدل مقداردهی اولیه میکند. مقادیر لحظهای نشاندهندهی گشتاور یا سرعت تغییر وزنها در الگوریتم بهینهسازی هستند. این تابع برای هر وزن و بایاس یک مقدار صفر را در لیستهای self.moms_w2, self.moms_b1, self.moms_w1 و self.moms_b2 ذخیره میکند.

• def _initalize_RMSs(self): این تابع مقادیر میانگین مربعات (RMS) را برای مدل مقداردهی اولیه میکند. مقادیر میانگین مربعات نشاندهندهی میزان تغییرات وزنها در الگوریتم بهینهسازی هستند. این تابع برای هر وزن و بایاس یک مقدار صفر را در لیستهای self.RMSs_w2, self.RMSs_b1, self.RMSs_w1 و self.RMSs_b2 ذخیره میکند.

• def _nn(self, xb): این تابع تابع پیشبینی مدل را تعریف میکند. این تابع یک تنسور به نام xb را به عنوان ورودی میگیرد و خروجی مدل را به عنوان خروجی میدهد. این تابع سه لایه را برای مدل ایجاد میکند: لایه اول یک لایه خطی است که ورودی را با وزنها و بایاسهای self.weights_1 و self.bias_1 ضرب میکند. لایه دوم یک تابع فعالسازی ReLU است که مقادیر منفی را صفر میکند. لایه سوم یک لایه خطی دیگر است که خروجی لایه دوم را با وزنها و بایاسهای self.weights_2 و self.bias_2 ضرب میکند. خروجی این لایه نهایی خروجی مدل است.

ین تابع train یک تابع است که برای آموزش یک مدل شبکه عصبی دو لایه با استفاده از یک بهینهساز دلخواه نوشته شده است. این تابع چهار نرخ یادگیری مختلف را امتحان میکند و نمودارهای تغییر خطای آموزش را برای هر یک رسم میکند. این تابع از چند مرحله تشکیل شده است:

• برای هر نرخ یادگیری، یک حلقه تکرار ایجاد میکند که تا زمانی که خطای آموزش کمتر از ۰٫۱ شود یا تعداد تکرار بیشتر از ۱۰۰۰ برسد ادامه مییابد.

• در هر تکرار، یک پیشبینی از مدل برای دادههای آموزشی محاسبه میشود و خطای آموزش با استفاده از تابع خطای مشخص شده محاسبه میشود.

• با استفاده از یک شیء GradientTape، گرادیانهای وزنهای قابل آموزش مدل نسبت به خطا را استخراج میکند.

• با استفاده از شیء بهینهساز، وزنهای مدل را با استفاده از گرادیانها و نرخ یادگیری بهروزرسانی میکند.

• خطای آموزش را در یک لیست ذخیره میکند.

• در پایان هر نرخ یادگیری، نمودار تغییر خطای آموزش را رسم میکند و پارامترهای مدل را مجدداً مقداردهی اولیه میکند.

مقایسه SGD و momentum:

بهینهساز SGD یا گرادیان کاهشی تصادفی یک روش مبتنی بر تکرار برای بهینهسازی یک تابع هدف است که در هر مرحله از یک تقریب تصادفی از گرادیان تابع استفاده میکند. بهینهساز Momentum یک تغییر در روش SGD است که از یک متغیر انباشته

برای ذخیره گرادیانهای گذشته استفاده میکند و در هر مرحله از ترکیبی از گرادیان فعلی و متغیر انباشته برای بهروزرسانی پارامترها استفاده میکند. این روش میتواند سرعت همگرایی را افزایش دهد و از گیر کردن در نقاط ثابت محلی جلوگیری کند.
مزایای Momentum نسبت به SGD عبارتند از:

- Momentum میتواند اثر نویز را کاهش دهد و به جهت بهینهتر حرکت کند.
- Momentum میتواند از گیر کردن در نقاط ثابت محلی یا صفحات مسطح خارج شود و به سمت نقاط ثابت سراسری پیش برود.
- Momentum میتواند سرعت یادگیری را تنظیم کند و از نوسانات زیاد در مسیر یادگیری جلوگیری کند.

تکانه سریعتر از نزول گرادیان تصادفی است و تمرین سریعتر از SGD خواهد بود.

حداقل های لوکال می تواند یک گریز باشد و به دلیل حرکت درگیر به حداقل های گلوبال برسد.

نرخ آموزش های مناسب برای این دو بهینه ساز:

همه ی نرخ های آموزش برای مومنتوم مناسب بودند اما برای SGD دو نرخ مناسب1 0.00 و 0.01 است و محدوده مقادیری که باید برای نرخ یادگیری در نظر گرفت کمتر از ۱,۰ و بیشتر از ۱۰^-۶ است. یک مقدار پیش فرض سنتی برای نرخ یادگیری ۰,۱ یا ۰,۰۱ است و این ممکن است نقطه شروع خوبی برای ماست.

اگر فقط زمان برای بهینه‌سازی یک هایپرپارامتر وجود داشته باشد و یکی از شیب نزولی تصادفی استفاده کند، این پارامتری است که ارزش تنظیم کردن را دارد. متأسفانه، نمی‌توانیم نرخ یادگیری بهینه را برای یک مدل معین در یک مجموعه داده معین به صورت تحلیلی محاسبه کنیم. در عوض، یک نرخ یادگیری خوب (یا به اندازه کافی خوب) باید از طریق آزمون و خطا کشف شود. محدوده مقادیری که باید برای نرخ یادگیری در نظر گرفت کمتر از ۱,۰ و بیشتر از ۱۰^-۶ است. یک مقدار پیش فرض سنتی برای نرخ یادگیری ۰,۱ یا ۰,۰۱ است و این ممکن است نقطه شروع خوبی برای مشکل شما باشد. رویکرد جستجوی شبکه‌ای می‌تواند به برجسته کردن مرتبه بزرگی که ممکن است نرخ یادگیری خوب وجود داشته باشد، کمک کند، و همچنین به توصیف رابطه بین نرخ یادگیری و عملکرد کمک کند.

اگر نرخ یادگیری ۱ در SGD باشد، ممکن است بسیاری از راه حل های کاندید را دور بریزید، و برعکس اگر بسیار کوچک باشد، ممکن است برای همیشه زمان ببرد تا راه حل مناسب یا راه حل بهینه را پیدا کنید. اگر نرخ یادگیری به اندازه کافی کوچک دارید، می توانید بر روی تخمین گرادیان به خوبی در تعدادی از تکرارها تکرار کنید.

اگر میزان یادگیری شما خیلی پایین تنظیم شده باشد، تمرینات بسیار آهسته پیشرفت خواهند کرد زیرا در حال به روز رسانی بسیار کوچک وزنه های شبکه خود هستید. با این حال، اگر میزان یادگیری شما بیش از حد بالا تنظیم شود، می تواند باعث ایجاد رفتار واگرای نامطلوب در عملکرد ضرر شما شود

برای مومنتوم نیز می توان گفت که برای شروع 0.99, 0.97, 0.95, and 0.9 نقاط خوبی برای شروع می توانند باشند

محور افقی اموزش های ما و عمودی لاس و خطای ما را براساس نرخ اموزش های مختلف نسان می دهد و در نرخ های مختلف مزیت ها و توانایی بیش تر مومنتوم در مقایسه با SGD مشهود می باشد

توضیحی بر کد های این دو :

```
def SGD(a, lr, _, __):
    a.data -= a.grad * lr
    a.grad = None
```

در یادگیری ماشین استفاده (SGD) این تابع یک تابع است که برای پیادهسازی الگوریتم بهینهسازی گرادیان کاهشی تصادفی میشود. این الگوریتم یک روش مبتنی بر تکرار برای بهینهسازی یک تابع هدف است که در هر مرحله از یک تقریب تصادفی از گرادیان تابع استفاده میکند. این تابع سه پارامتر را دریافت میکند:

- a: یک پارامتر قابل آموزش مدل، مانند وزن یا بایاس

- lr: یک نرخ یادگیری، که میزان تغییر پارامتر را در هر مرحله تعیین میکند

- _: که در این تابع استفاده نمیشودa یک لیست از متغیرهای انباشته برای پارامتر ،

این تابع دو مرحله را انجام میدهد:

- و نرخ یادگیری بهروزرسانی میکند a را با کم کردن حاصل ضرب گرادیان فعلی پارامتر a پارامتر

- را صفر میکند a گرادیان پارامتر

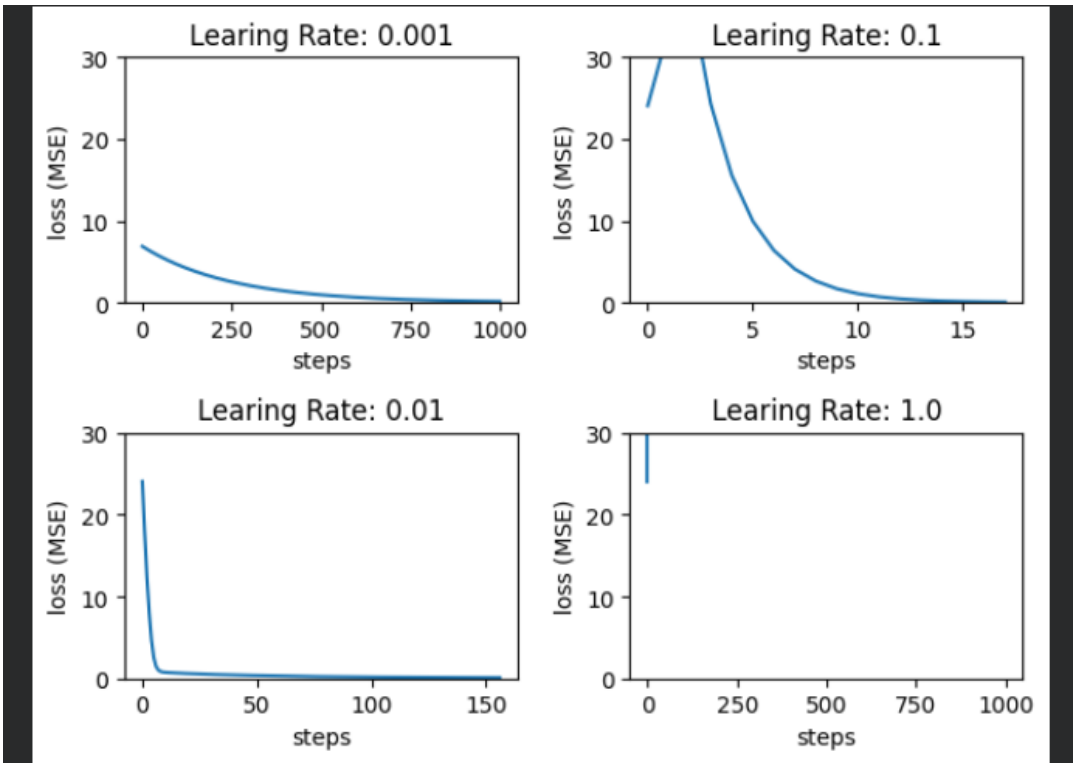
```
def momentum(a, _, moms, __):
    previous_momentum = moms[-1]

    mom = a.grad * 0.1 + previous_momentum * 0.9
    moms.append(mom)
    a.data -= mom
    a.grad = None
```

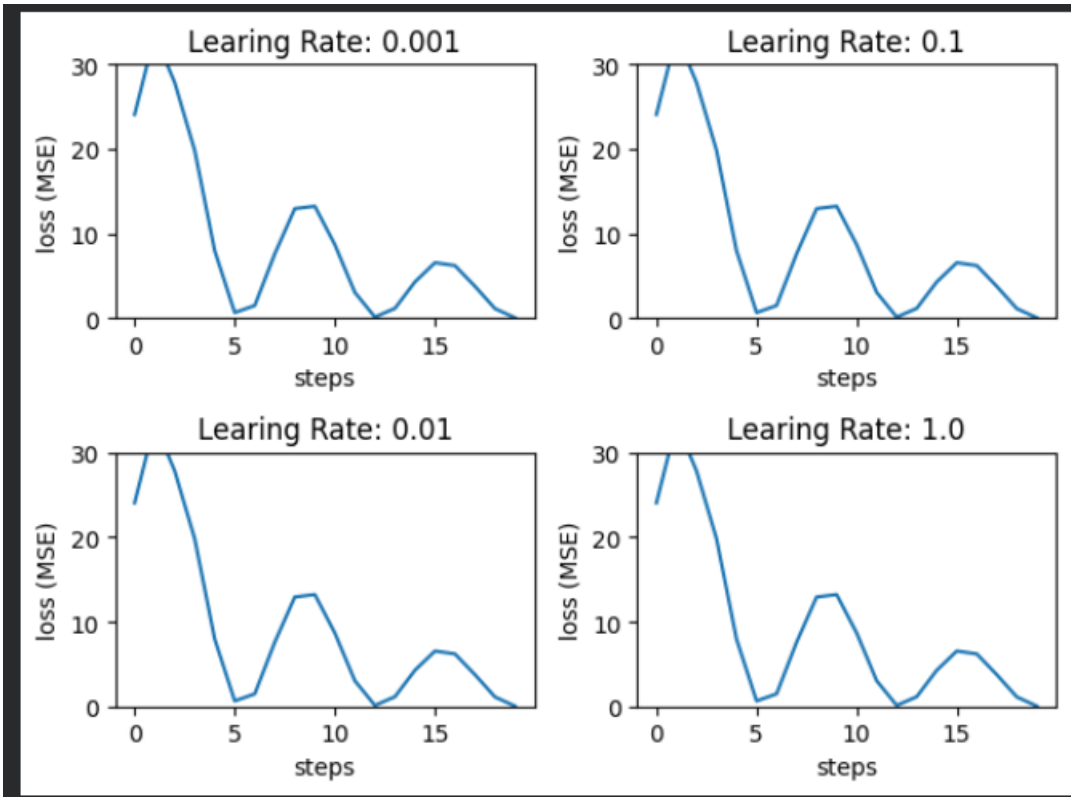
ین تابع یک تابع است که برای پیادهسازی الگوریتم بهینهسازی momentum در یادگیری ماشین استفاده میشود. این الگوریتم یک تکنیک برای بهبود عملکرد گرادیان کاهشی تصادفی (SGD) است که از یک متغیر انباشته برای ذخیره گرادیانهای گذشته استفاده میکند. این متغیر با یک ضریب تنزلی (decay factor) ضرب شده و با گرادیان فعلی جمع میشود. این کار باعث میشود که الگوریتم سرعت بیشتری داشته باشد و از گیر کردن در حداقلهای محلی خارج شود. این تابع چهار پارامتر را دریافت میکند:

- a: یک پارامتر قابل آموزش مدل، مانند وزن یا بایاس
- _: یک نرخ یادگیری، که میزان تغییر پارامتر را در هر مرحله تعیین میکند
- moms: یک لیست از متغیرهای انباشته برای پارامترa ، که در هر مرحله به روزرسانی میشود
- ___: یک لیست از متغیرهای انباشته برای مربع گرادیان پارامترa ، که در این تابع استفاده نمیشود

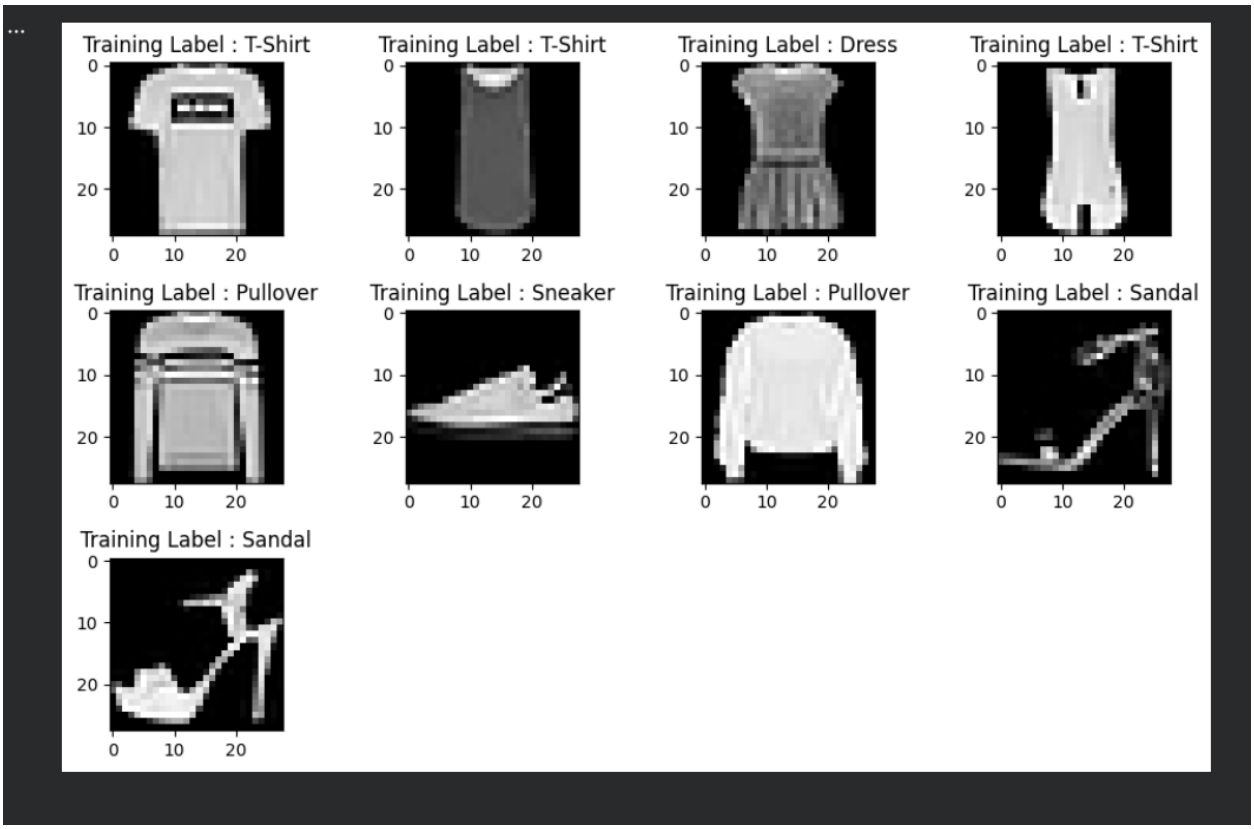
این تابع چند مرحله را انجام میدهد:

- متغیر انباشته قبلی را از لیست moms دریافت میکند
- متغیر انباشته جدید را با استفاده از گرادیان فعلی پارامتر a و متغیر انباشته قبلی محاسبه میکند
- متغیر انباشته جدید را به لیست moms اضافه میکند
- پارامتر a را با کم کردن متغیر انباشته جدید بهروزرسانی میکند
- گرادیان پارامتر a را صفر میکند


Learing Rate: 0.001
loss (MSE)
steps
Learing Rate: 0.1
loss (MSE)
steps
Learing Rate: 0.01
loss (MSE)
steps
Learing Rate: 1.0
loss (MSE)
steps


Learing Rate: 0.001
Learing Rate: 0.1
Learing Rate: 0.01
Learing Rate: 1.0
loss (MSE)
steps
0
5
10
15
20
30

سوال ۵)


Training Label : T-Shirt
Training Label : T-Shirt
Training Label : Dress
Training Label : T-Shirt
Training Label : Pullover
Training Label : Sneaker
Training Label : Pullover
Training Label : Sandal
Training Label : Sandal

```
    ################ Your code ############
    model = nn.Sequential(
    torch.nn.Flatten(),
    torch.nn.Linear(784, 128),
    torch.nn.ReLU(),
    torch.nn.Linear(128, 128),
    torch.nn.ReLU(),
    torch.nn.Linear(128, 10),
    torch.nn.LogSoftmax(dim=1)

    )
    ########################################
```

```
    ################ Your code ############

    criterion = torch.nn.CrossEntropyLoss()
    optimizer =optim.SGD( model.parameters(), lr=0.01)
    ########################################
```

تعداد نورون ها و لایه ها در هر لایه

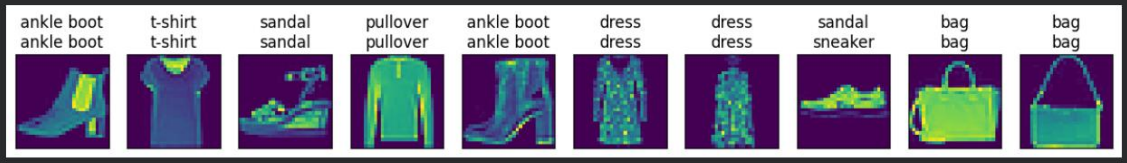
```
    print(model)
```

```
Sequential(
  (0): Flatten(start_dim=1, end_dim=-1)
  (1): Linear(in_features=784, out_features=128, bias=True)
  (2): ReLU()
  (3): Linear(in_features=128, out_features=128, bias=True)
  (4): ReLU()
  (5): Linear(in_features=128, out_features=10, bias=True)
  (6): LogSoftmax(dim=1)
)
```

```
!pip install --upgrade d2l==0.17.0
```

```
Requirement already satisfied: d2l==0.17.0 in /usr/local/lib/python3.10/dist-packages (0.17.0)
Requirement already satisfied: jupyter in /usr/local/lib/python3.10/dist-packages (from d2l==0.17.0) (1.0.0)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from d2l==0.17.0) (1.23.5)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist-packages (from d2l==0.17.0) (3.7.2)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from d2l==0.17.0) (2.31.0)
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packages (from d2l==0.17.0) (2.0.3)
Requirement already satisfied: notebook in /usr/local/lib/python3.10/dist-packages (from jupyter->d2l==0.17.0) (6.5.5)
Requirement already satisfied: qtconsole in /usr/local/lib/python3.10/dist-packages (from jupyter->d2l==0.17.0) (5.5.0)
Requirement already satisfied: jupyter-console in /usr/local/lib/python3.10/dist-packages (from jupyter->d2l==0.17.0) (6.1.0)
Requirement already satisfied: nbconvert in /usr/local/lib/python3.10/dist-packages (from jupyter->d2l==0.17.0) (6.5.4)
Requirement already satisfied: ipykernel in /usr/local/lib/python3.10/dist-packages (from jupyter->d2l==0.17.0) (5.5.6)
```

```
## Test your model
from d2l import torch as d2l
d2l.predict_ch3(model,testloader,n = 10)
```



توضیحی بر کد :

این کد از دیتاست FashionMNIST استفاده میکند که شامل ۶۰۰۰۰ تصویر آموزشی و ۱۰۰۰۰ تصویر آزمون از ۱۰ کلاس مختلف لباس است. این کد شامل چند بخش است:

- وارد کردن کتابخانههای مورد نیاز
- تعریف پارامترهای مدل، مانند اندازه ورودی، اندازه خروجی، نرخ دراپاوت و نرخ یادگیری
- بارگذاری دیتاست و تبدیل آن به تنسورهای PyTorch
- تعریف کلاسهای دیتاست، مانند T-Shirt, Trouser, Pullover و غیره
- تعریف مدل شبکه عصبی چند لایه با استفاده از کلاس nn.Sequential و اضافه کردن لایههای با احتمال مشخص
- تعریف تابع هزینه و بهینهساز

- آموزش مدل بر روی دادههای آموزشی با استفاده از حلقههای تکرار و بهروزرسانی پارامترها

- آزمون مدل بر روی دادههای آزمون و محاسبه دقت مدل

تعداد لایه ها و نورن ها و دلیل انتخاب :

سعی کردم کمترین تعداد لایه های لازم را انتخاب کنم یه لایه وردی یه لایه خروجی و یه لایه پنهان

تعداد نورون های خروجی به اندازه کلاس ها بوده است و تعداد وذودی ها به اندازه پیکسل های تصویر همچنین از توابع فعال ساز استفاده شده در استخراج ویژگی ها و مدل بهینه تر کمک کنند

تعداد نورون های لایه ی پنهان هم ۱۲۸ انتخاب شده کمتر از ورودی باشد و توانایی یادگیری خوبی را داشته باشد تا مدل به خوبی ترین وتست شود

- یک لایه Flatten که تصویر ورودی را با ابعاد x28۲۸ پیکسل به یک بردار یک بعدی با اندازه ۷۸۴ تبدیل می کند.

- یک لایه Linear که بردار ورودی را به یک بردار خروجی با اندازه ۱۲۸ تبدیل می کند. این لایه همچنین یک وزن و یک بایاس دارد که در طول آموزش بهینه می شوند.

- یک لایه ReLU که یک تابع فعال سازی غیر خطی است و عناصر منفی بردار ورودی را صفر می کند.

- یک لایه Linear دیگر که بردار ورودی را به یک بردار خروجی با اندازه ۱۲۸ تبدیل می کند. این لایه نیز یک وزن و یک بایاس دارد که در طول آموزش بهینه می شوند.

- یک لایه ReLU دیگر که مانند قبل عمل می کند.

- یک لایه Linear آخر که بردار ورودی را به یک بردار خروجی با اندازه ۱۰ تبدیل می کند. این لایه نماینده ۱۰ کلاس مختلف اعداد از ۰ تا ۹ است. این لایه نیز یک وزن و یک بایاس دارد که در طول آموزش بهینه می شوند.

- یک لایه LogSoftmax که یک تابع فعال سازی خطی است و بردار ورودی را به یک بردار احتمالاتی تبدیل می کند.

- تجربه و دانش فردی که شبکه عصبی را طراحی میکند. برای مثال، برای انتخاب بهترین تابع فعالسازی، تعداد نورونها، نرخ یادگیری و دیگر پارامترهای شبکه عصبی، معمولا از دانش نظری و تجربی در زمینهی یادگیری ماشین و شبکههای عصبی استفاده میشود.

دلیل استفاده از رلو

ایه های کانولوشن و یادگیری عمیق: محبوب ترین تابع فعال سازی برای آموزش لایه های کانولوشن و مدل های یادگیری عمیق است .سادگی محاسباتی: تابع یکسو کننده برای پیاده سازی بی اهمیت است و فقط به تابع max() نیاز دارد .پراکندگی نمایشی: یک مزیت مهم تابع یکسو کننده این است که قادر به خروجی یک مقدار صفر واقعی است .رفتار خطی: بهینه سازی یک شبکه عصبی زمانی که رفتار آن خطی یا نزدیک به خطی باشد آسان تر است

دلیل استفاده از softmax

تابع فعال‌سازی softmax با آسان‌تر کردن تفسیر خروجی‌های شبکه عصبی، این کار را برای شما ساده می‌کند !تابع فعال‌سازی softmax خروجی‌های خام شبکه عصبی را به بردار احتمالات تبدیل می‌کند، که اساساً یک توزیع احتمال روی کلاس‌های ورودی است.

نتیجه :

دیدیم که در مرحله ی ازمایش همه ی موارد را با لیبل هایسان درست پیش بینی کرده است و مدل بهخوبی کار می کند دستور اینستال dl2 هم در بالا اورده شده

سوال ۶)

ب)برای اینکه اورفیتینگ داشته باشیم تعداد نورون های ان را زیاد کنیم تا مدل پیچیده شود و دیتاها ثابت است و دیتای جدید نداریم واریانس زیاد شود و باعث اورفیتیگ شود و یا تعداد لایه را زیاد کنیم و یا ویژگی های بسیاری به مدل اضافه کنیم . و یا در تعداد ایپوک های بیش تری مدل را اموزش دهیم تا دچار اورفیت بشود.حالا فقط حفظ کرده و روی تست نتایج خوبی ندارد.

تعداد ایپاک ها را بسیار زیاد کردیم =۶۰

و تعدا نورون ها را هم به مقدار قابل توجهی زیاد کردیم ۵۱۲ و

مدل دچار اورفیت بسیار شدید یشد و مقدار داده ترین و تست از لحاظ خطا با هم اختلاف فاحسی دارند .

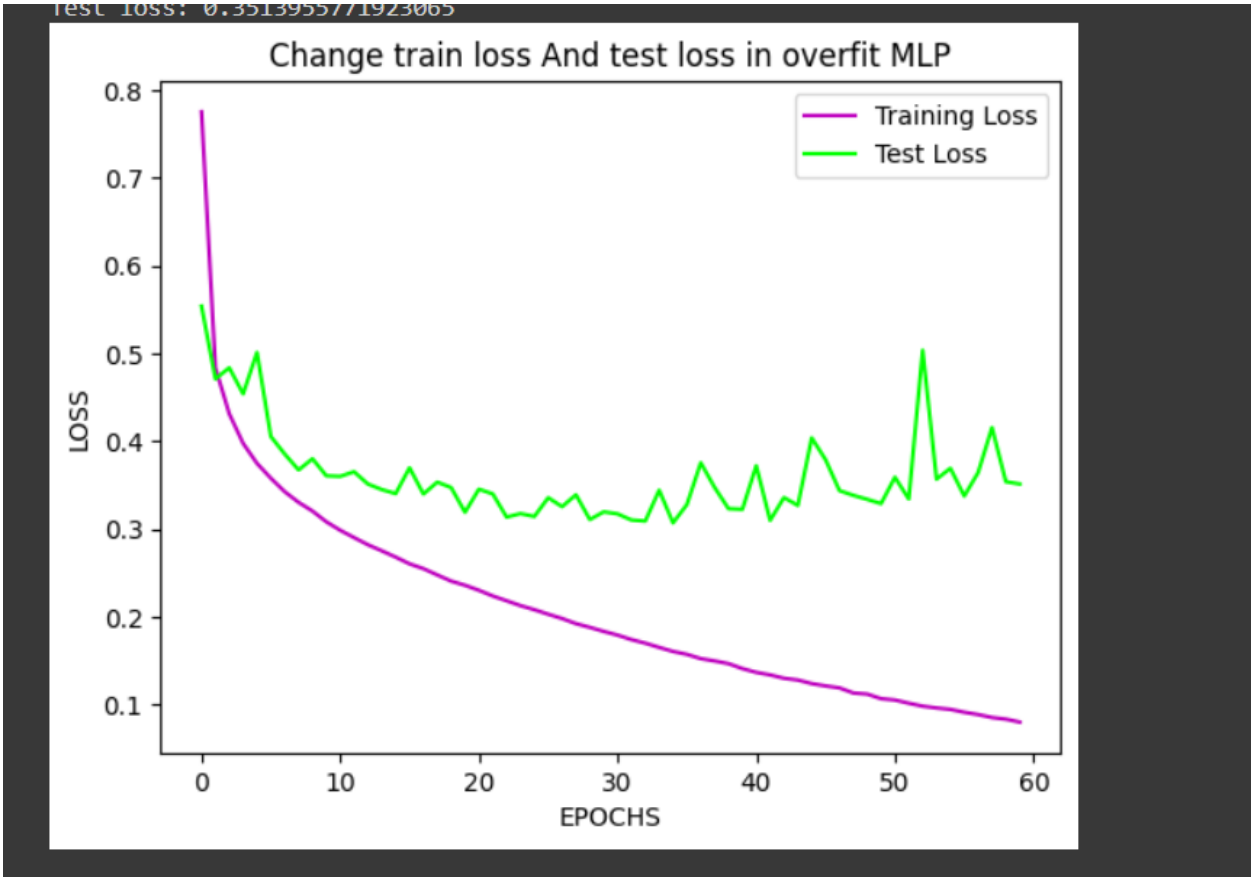
```python
model_overfit = nn.Sequential(
torch.nn.Flatten(),
torch.nn.Linear(784, 1024),
torch.nn.ReLU(),
torch.nn.Linear(1024, 512),
torch.nn.ReLU(),
torch.nn.Linear(512, 10),
torch.nn.LogSoftmax(dim=1)

)
print(model_overfit)
```

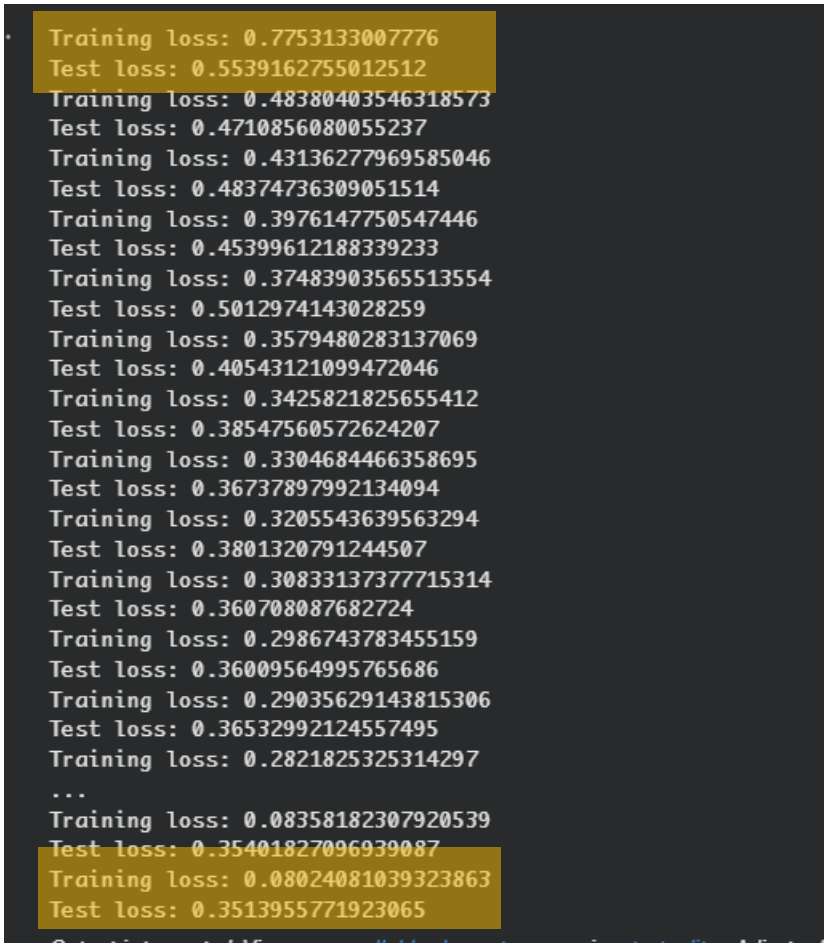
```
Sequential(
  (0): Flatten(start_dim=1, end_dim=-1)
  (1): Linear(in_features=784, out_features=1024, bias=True)
  (2): ReLU()
  (3): Linear(in_features=1024, out_features=512, bias=True)
  (4): ReLU()
  (5): Linear(in_features=512, out_features=10, bias=True)
  (6): LogSoftmax(dim=1)
)
```

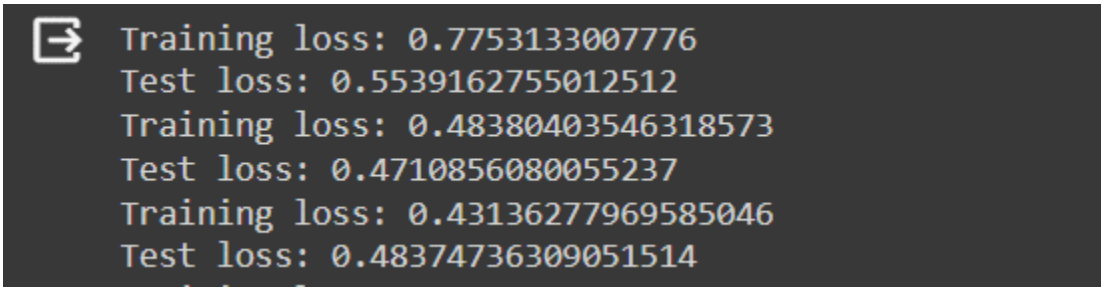
افزایش قابل توجه نورون ها

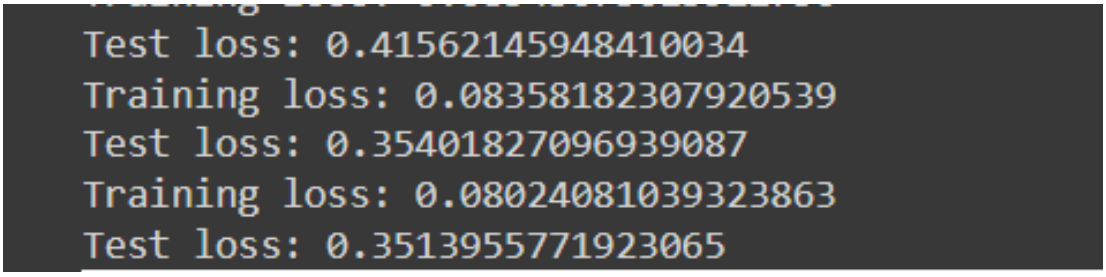
```python
for e in range(60):
    running_loss = 0
    for images, labels in trainloader:
        images = images.view(images.shape[0], -1)

        optimizer.zero_grad()

        output = model_overfit(images)
        loss = criterion(output, labels)

        loss.backward()
        optimizer.step()

        running_loss += loss.item()
    train_losses.append(running_loss / len(trainloader))
    print(f"Training loss: {running_loss/len(trainloader)}")
```

و رساندن مراحل اموزش به ۶۰ تا داده ها را حفظ کند و رو همان ها فقط فیت شود و درنتیجه مقدار بسیار زیاد لاس داده های تست را در مقابل داده های اموزسی مشاهده می کنیم یا همان اورفیتینگ




Training loss: 0.7753133007776
Test loss: 0.5539162755012512
Training loss: 0.48380403546318573
Test loss: 0.4710856080055237
Training loss: 0.43136277969585046
Test loss: 0.48374736309051514
Training loss: 0.3976147750547446
Test loss: 0.45399612188339233
Training loss: 0.37483903565513554
Test loss: 0.5012974143028259
Training loss: 0.3579480283137069
Test loss: 0.40543121099472046
Training loss: 0.3425821825655412
Test loss: 0.38547560572624207
Training loss: 0.3304684466358695
Test loss: 0.36737897992134094
Training loss: 0.3205543639563294
Test loss: 0.3801320791244507
Training loss: 0.30833137377715314
Test loss: 0.360708087682724
Training loss: 0.2986743783455159
Test loss: 0.36009564995765686
Training loss: 0.29035629143815306
Test loss: 0.36532992124557495
Training loss: 0.2821825325314297
...
Training loss: 0.08358182307920539
Test loss: 0.35401827096939087
Training loss: 0.08024081039323863
Test loss: 0.3513955771923065


Training loss: 0.7753133007776
Test loss: 0.5539162755012512
Training loss: 0.48380403546318573
Test loss: 0.4710856080055237
Training loss: 0.43136277969585046
Test loss: 0.48374736309051514


Test loss: 0.41562145948410034
Training loss: 0.08358182307920539
Test loss: 0.35401827096939087
Training loss: 0.08024081039323863
Test loss: 0.3513955771923065

پ)داده فزایی

درمورد قسمت دیتا اگمنتیشن:

درواقع وقتی دیتاهامون کمه، میایم یکم دیتاهامونو تغییر میدیم و چندتا ازش میسازیم، مثلا فک کن میخوایم تشخیص گربه بدیم و کلا ده تا عکس گربه داریم، برا اینکه تعداد دیتاهامون رو بیشتر کنیم مثلا میایم رو هرکدوم از این گربه ها فیلتر میزاریم (اینجوری تعداد عکسای گربمون بیشتر میشه اگرچه عکس همون عکسه تقریبا) یا مثلا میایم یه قسمتایی از تصویر رو کراپ میکنیم و میبریم در واقع با این تکنیک میایم و تعداد دادههای اولیمون رو بیشتر میکنیم مثلا اگه اولش ده تا داده داشتیم با این کارا همون ده تا رو تبدیل به هزار تا دیتا میکنیم

بعد اینجوری تو داده های اموزشیت بیشتر میشه، در نتیجه این تو میتونی میزان واریانس رو کم کنی و درنتیجه اورفیت شدن رو کم کنی

مثلا اولیش که جئومتریکه گفته بصورت رندوم مثلا میایم عکس رو آینه وار فلیپ میکنیم یا یه قسمتایی از تصویرو پاک میکنیم و یا مثلا روی رنگ های تصویر تغییر میدیم (مثلا سیاه سفید میکنیم یا رنگشونو تغییر میدیم)

توی این لینک که توی خود داکیومنت تمرینت گذاشته هم اومده گفته ما یه کلاسی توی پای تورچ ساختیم به اسم transform و به کمک این میتونیم بگیم چیکارا روی تصویر انجام بده و بعد تصاویر رو به این ترنسفورمه میدیم که اون تغییرات رو بده

https://pytorch.org/vision/master/transforms.html

اومده اول عکس رو تبدیل کرده به تنسور بعد به ترتیب اومده کد رنگ هارو تبدیل کرده به عدد صحیح، بعد به کمک تابع RandomResizedCrop اومده بصورت رندوم یه جای تصویر رو برش زده (اینجوری تصویر جدید خلق کرده) و بعدم به همین منوال دیتاهارو نرمالایز کرده (از حالت عدد صحیح)

بعد اینجوری تو داده های اموزشیت بیشتر میشه، در نتیجه این تو میتونی میزان واریانس رو کم کنی و درنتیجه اورفیت شدن رو کم کنیبعد وقتی تو همین لینکه بیای پایین توی این قسمت اومده قانکشنای مختلفی که میتونه رو عکس انجام بده رو گفته (مثلا کوچیک و بزرگ کردن تصویر یا برش زدن تصویر یا چرخش تصویر یا مثلا زوم کردن روی تصویر یا تغییر رنگای تصویر و...)

توضیح کد و توضیح اینکه از چه مواردی استفاده شده:

نتیجه :

به خوبی مقدار اورفیتیگ کاهش یافته و داده های تست هم به خوبی عمل کرده اند و حتی مقدار لاس ان ها از داده های اموزشی در برخی نقاط کم تر شده و لاس قابل قبولی را دارند در نتیجه با افزودن دیتاهای مختلف توانستیم اورفیتینگ را کم کنیم.

```
import torchvision.transforms as transforms
transform_train = transforms.Compose([
    transforms.RandomRotation(10),
    transforms.RandomHorizontalFlip(),
    transforms.RandomVerticalFlip(),
    transforms.ToTensor(),
])

transform_test = transforms.Compose([
    transforms.ToTensor(),
])

trainset = FashionMNIST(root='./data', train=True, download=True, transform=transform_train)
trainloader = DataLoader(trainset, batch_size=64, shuffle=True)

testset = FashionMNIST(root='./data', train=False, download=True, transform=transform_test)
testloader = DataLoader(testset, batch_size=64, shuffle=False)

train_losses = []
test_losses = []
```

این کد چند مرحله را برای آماده سازی و بارگذاری داده های مجموعه داده FashionMNIST انجام می دهد. این مجموعه داده شامل ۷۰ هزار تصویر سیاه و سفید از لباس ها و کفش های مختلف است که به ۱۰ کلاس تقسیم شده اندhttps://pytorch.org/vision/stable/generated/torchvision.datasets.FashionMNIST.html. این کد از بسته torchvision.transforms استفاده می کند که شامل چندین تابع برای تغییر و افزایش داده های تصویری استhttps://pytorch.org/tutorials//beginner/basics/transforms_tutorial.html. این کد از توابع زیر استفاده می کند:

- transforms.Compose: یک تابع که چندین تبدیل را در یک لیست قرار می دهد و یک تبدیل جدید ایجاد می کند که تمام تبدیل های لیست را به ترتیب اعمال می کندhttps://pytorch.org/tutorials//beginner/basics/transforms_tutorial.html.

- transforms.RandomRotation: یک تابع که تصویر را با یک زاویه تصادفی در بازه داده شده چرخانده می کندhttps://pytorch.org/tutorials//beginner/basics/transforms_tutorial.html.

- transforms.RandomHorizontalFlip: یک تابع که تصویر را با احتمال ۰٫۵ افقی معکوس می کندhttps://pytorch.org/tutorials//beginner/basics/transforms_tutorial.html.

- transforms.RandomVerticalFlip: یک تابع که تصویر را با احتمال ۰٫۵ عمودی معکوس می کندhttps://pytorch.org/tutorials//beginner/basics/transforms_tutorial.html.

- transforms.ToTensor: یک تابع که تصویر را از یک شیء PIL یا numpy array به یک تنسور pytorch تبدیل می کند

این کد دو تبدیل مختلف برای داده های آموزش و آزمون ایجاد می کند. تبدیل آموزش شامل چرخش، معکوسی افقی و عمودی تصادفی است که برای افزایش تنوع داده ها و جلوگیری از بیش برازش مفید است. تبدیل آزمون فقط شامل تبدیل تصویر به تنسور است که برای سازگاری با مدل لازم است.

این کد همچنین از کلاس FashionMNIST برای بارگذاری مجموعه داده استفاده می کند. این کلاس یک زیر کلاس از کلاس Dataset است که یک رابط برای دسترسی به داده ها را فراهم می کندl. این کلاس چند پارامتر را می پذیرد:

- root: مسیر پوشه ای که داده ها در آن ذخیره می شوند

- train: یک مقدار بولی که مشخص می کند که داده های آموزش یا آزمون بارگذاری شوند

- download: یک مقدار بولی که مشخص می کند که آیا داده ها از اینترنت دانلود شوند یا خیرhtt

- transform: یک تابع که بر روی تصاویر اعمال

این کد همچنین از کلاس DataLoader برای تقسیم داده ها به دسته های کوچکتر و قابل مدیریت تر استفاده می کند. این کلاس یک ابزار برای بارگذاری داده ها به صورت موازی و کارآمد است. این کلاس چند پارامتر را می پذیرد:

- dataset: یک شیء Dataset که داده ها را نگه می دارد.

- batch_size: تعداد داده هایی که در هر دسته قرار می گیرند.

- shuffle: یک مقدار بولی که مشخص می کند که آیا داده ها به صورت تصادفی قبل از بارگذاری مخلوط شوند یا خیر.

نحوه ی تغییرات از اولین تا اخرین مقدار که به خوبی بیانگر این است که بیش برازش هندل شده

افزایش داده ها تکنیکی برای افزایش مصنوعی مجموعه آموزشی با ایجاد نسخه های اصلاح شده از یک مجموعه داده با استفاده از داده های موجود است. این شامل ایجاد تغییرات جزئی در مجموعه داده یا استفاده از یادگیری عمیق برای تولید نقاط داده جدید است.

برای جلوگیری از برازش بیش از حد مدل ها.

مجموعه آموزشی اولیه خیلی کوچک است.

برای بهبود دقت مدل

برای کاهش هزینه عملیاتی برچسب زدن و تمیز کردن مجموعه داده خام.

تقویت تصویر

در مسیر مهارت پردازش تصویر با پایتون، درباره تغییر شکل و دستکاری تصویر با تمرینات عملی بیشتر بیاموزید.

تغییرات هندسی: به طور تصادفی ورق زدن، برش، چرخش، کشش و بزرگنمایی تصاویر. باید مراقب اعمال چندین تغییر شکل روی تصاویر مشابه باشید، زیرا این کار می تواند عملکرد مدل را کاهش دهد.

تغییر فضای رنگ: به طور تصادفی کانال های رنگ، کنتراست و روشنایی RGB را تغییر دهید.

فیلترهای هسته: به طور تصادفی وضوح یا تاری تصویر را تغییر می دهند.

پاک کردن تصادفی: بخشی از تصویر اولیه را حذف کنید.

اختلاط تصاویر: ترکیب و ترکیب چندین تصویر.

RANDOMVERTICALFLIP

Vertically flip the given image randomly with a given probability. If the image is torch Tensor, it is expected to have […, H, W] shape, where … means an arbitrary number of leading dimensions

Parameters:

p (*float*) – probability of the image being flipped. Default value is 0.5

transforms.RandomRotation(10), **RANDOMROTATION**

CLASStorchvision.transforms.RandomRotation(*degrees*, *interpolation=InterpolationMode.NEAREST*, *expand=False*, *center=None*, *fill=0*)[SOURCE]

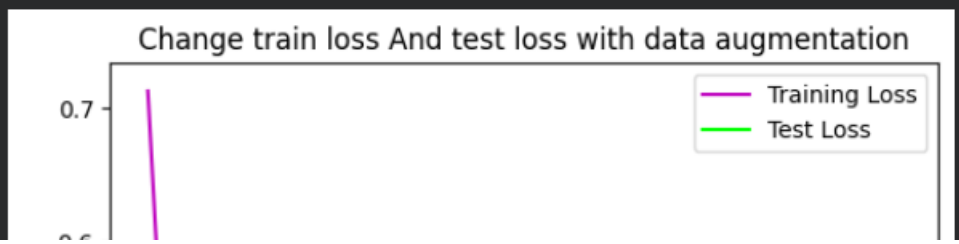
Rotate the image by angle. If the image is torch Tensor, it is expected to have […, H, W] shape, where … means an arbitrary number of leading dimensions.

Parameters:

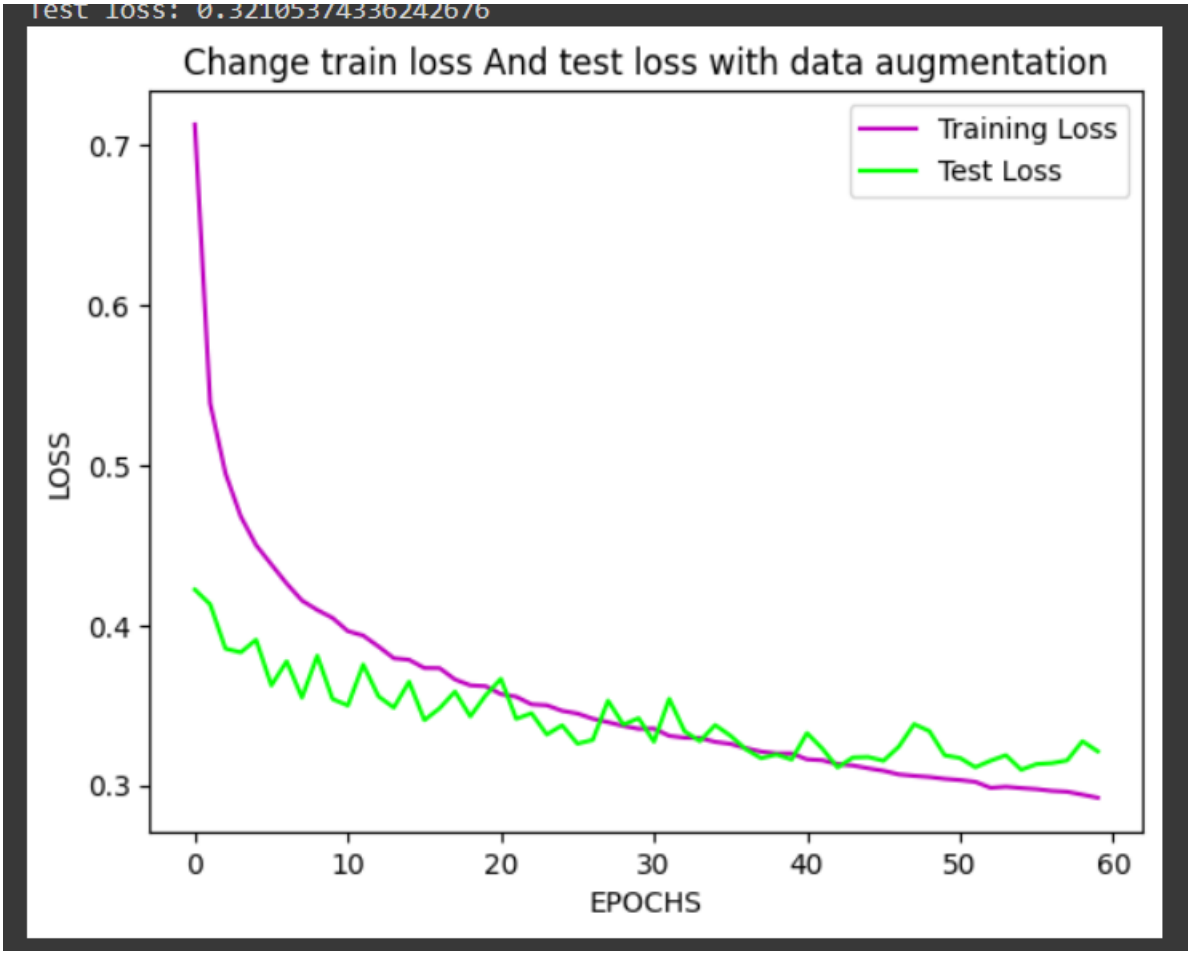
- **degrees** (*sequence or number*) – Range of degrees to select from. If degrees is a number instead of sequence like (min, max), the range of degrees will be (-degrees, +degrees).
- **interpolation** (*InterpolationMode*) – Desired interpolation enum defined by `torchvision.transforms.InterpolationMode`. Default is `InterpolationMode.NEAREST`. If input is Tensor, only `InterpolationMode.NEAREST`, `InterpolationMode.BILINEAR` are supported. The corresponding Pillow integer constants, e.g. `PIL.Image.BILINEAR` are accepted as well.
- **expand** (*bool, optional*) – Optional expansion flag. If true, expands the output to make it large enough to

```
Training loss: 0.7132520186685042
Test loss: 0.4222601652145386
Training loss: 0.5388090954914785
Test loss: 0.4130410850048065
Training loss: 0.49499903435010645
Test loss: 0.3853286802768707
Training loss: 0.46803387132153584
Test loss: 0.382962167263031
Training loss: 0.45008008801606675
Test loss: 0.39085498452186584
Training loss: 0.43799059331289997
Test loss: 0.3622870445251465
Training loss: 0.42604483249408603
Test loss: 0.37739989161491394
Training loss: 0.41540036974812367
Test loss: 0.3546847701072693
Training loss: 0.4095068276881663
Test loss: 0.3808952271938324
Training loss: 0.40453605194971254
Test loss: 0.3537795841693878
Training loss: 0.39625286485658273
Test loss: 0.3497578203678131
Training loss: 0.3935129884590726
Test loss: 0.37532514333724976
Training loss: 0.38665436169327194
...
Training loss: 0.29395620526471883
Test loss: 0.3273802101612091
Training loss: 0.29202617266411973
Test loss: 0.32105374336242676
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...



Test loss: 0.32105374336242676




```
Training loss: 0.7132520186685042
Test loss: 0.4222601652145386
Training loss: 0.5388090954914785
Test loss: 0.4130410850048065
Training loss: 0.49499903435010645
Test loss: 0.3853286802768707
Training loss: 0.46803387132153584
Test loss: 0.382962167263031
Training loss: 0.45008008801606675
Test loss: 0.39085498452186584
Training loss: 0.43799059331289997
Test loss: 0.3622870445251465
Training loss: 0.42604483249408603
Test loss: 0.37739989161491394
```

```
Test loss: 0.3095241189002991
Training loss: 0.2974057452145543
Test loss: 0.31306055188179016
Training loss: 0.2963194532006153
Test loss: 0.313647985458374
Training loss: 0.2957809068032229
Test loss: 0.3153814673423767
Training loss: 0.29395620526471883
Test loss: 0.3273802101612091
Training loss: 0.29202617266411973
Test loss: 0.32105374336242676
```

ت)منظم سازیL2 ,

https://stackoverflow.com/questions/42704283/l1-l2-regularization-in-pytorch

در این قسمت سه مدل را بررسی کردم ابتداا مدلی که به سادگی ۳۰ ایپاک را ترین و تست کرده و بعد مدل ساده را با منظم سازی و مدل اورفیت شده را با روش l2 بررسی کردم

```
optimizer = optim.SGD(model.parameters(), lr=0.01, weight_decay=1e-5)
```

این الگوریتم بهینه سازی SGD نام دارد و یکی از روش های ساده و معروف یادگیری عمیق است. این الگوریتم در هر مرحله یک قدم کوچک به سمت جهت منفی گرادیان می گیرد تا به حداقل تابع هزینه برسد. برای اطلاعات بیشتر می توانید به [این لینک] مراجعه کنید.

این کد چند پارامتر را به الگوریتم بهینه سازی می دهد:

- model.parameters(): یک تابع که یک تکرار شونده از تمام پارامترهای قابل بهینه سازی مدل را بر می گرداند. این پارامترها شامل وزن ها و بایاس های لایه های خطی و دیگر لایه هایی هستند که پارامتر دارند.

- lr: یک پارامتر که نرخ یادگیری را مشخص می کند. این پارامتر مقدار قدم را کنترل می کند و باید به طور دقیق انتخاب شود. اگر نرخ یادگیری خیلی بزرگ باشد، ممکن است الگوریتم بهینه سازی از حداقل تابع هزینه عبور کند و نتیجه خوبی ندهد. اگر نرخ یادگیری خیلی کوچک باشد، ممکن است الگوریتم بهینه سازی خیلی آهسته پیش رود و زمان زیادی برای رسیدن به حداقل تابع هزینه طول بکشد. مقدار پیش فرض این پارامتر ۰,۰۱ است.

• weight_decay: یک پارامتر که مقدار تنزل وزن را مشخص می کند. این پارامتر برای جلوگیری از بیش برازش مفید است. بیش برازش زمانی رخ می دهد که مدل بر روی داده های آموزش خیلی خوب عمل کند اما بر روی داده های جدید خوب عمل نکند. این پارامتر باعث می شود که وزن های مدل کمی کاهش یابند و مدل ساده تر و کلی تر شود. مقدار پیش فرض این پارامتر صفر است.

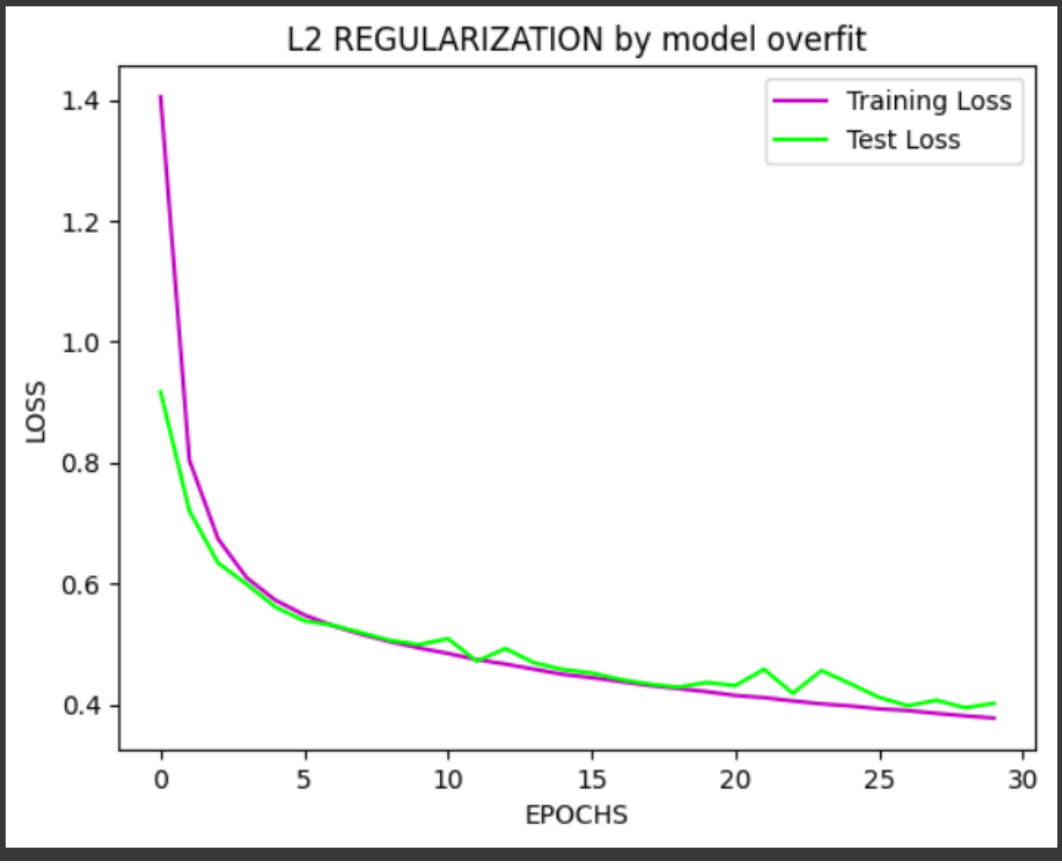
نظم‌سازی L2 (رگرسیون ریج) - مجموع مربع‌های همه وزن‌ها را در مدل به تابع هزینه اضافه می‌کند. این می تواند الگوهای داده پیچیده را یاد بگیرد و بر خلاف تنظیم L1 راه حل های غیر پراکنده ارائه می دهد.

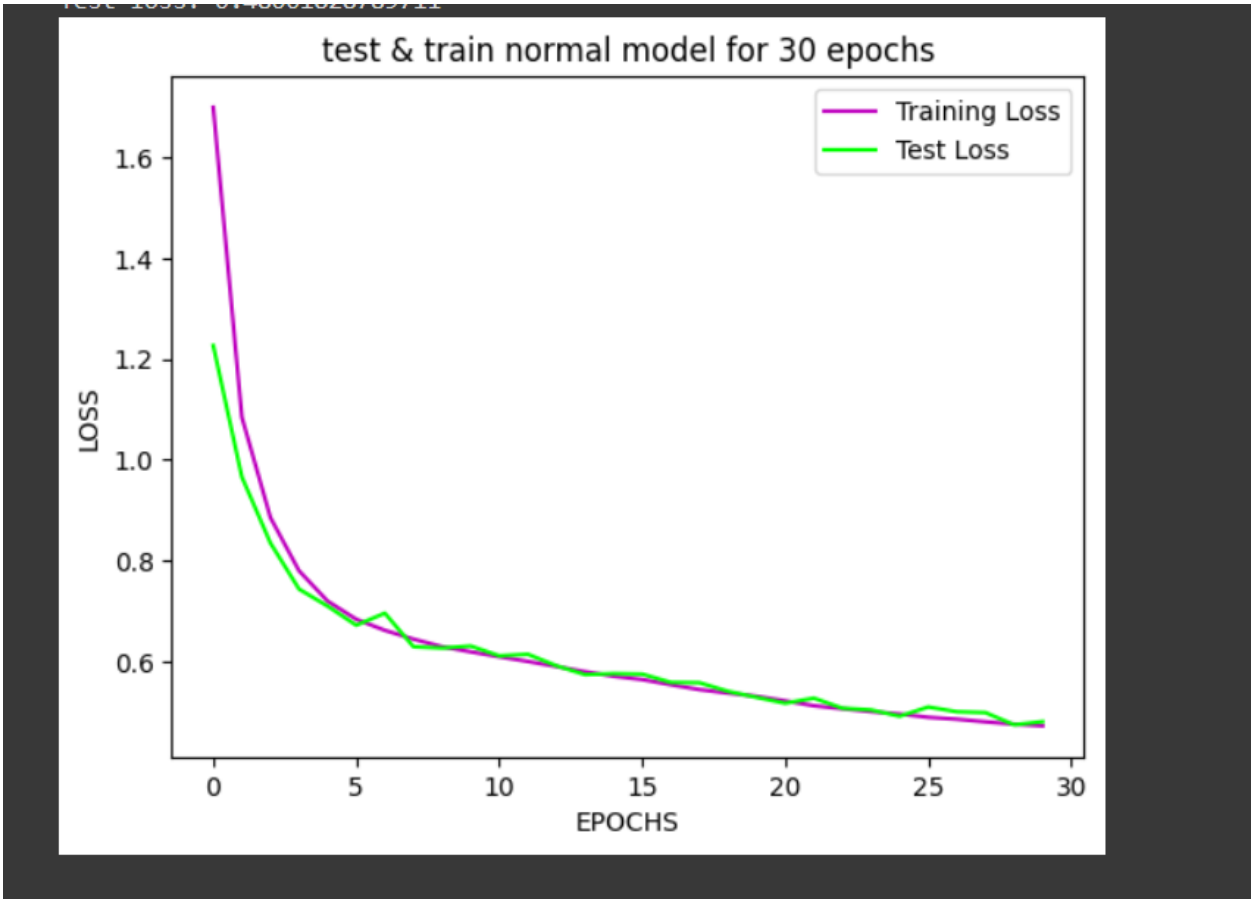
ب) تنظیم L2

پارامتر weight_decay تنظیم L2 را در حین شروع بهینه ساز اعمال می کند. این عبارت منظم‌سازی را به تابع ضرر اضافه می‌کند، با اثر کوچک کردن تخمین‌های پارامتر، مدل را ساده‌تر می‌کند و احتمال اضافه‌برازش را کمتر می‌کند.

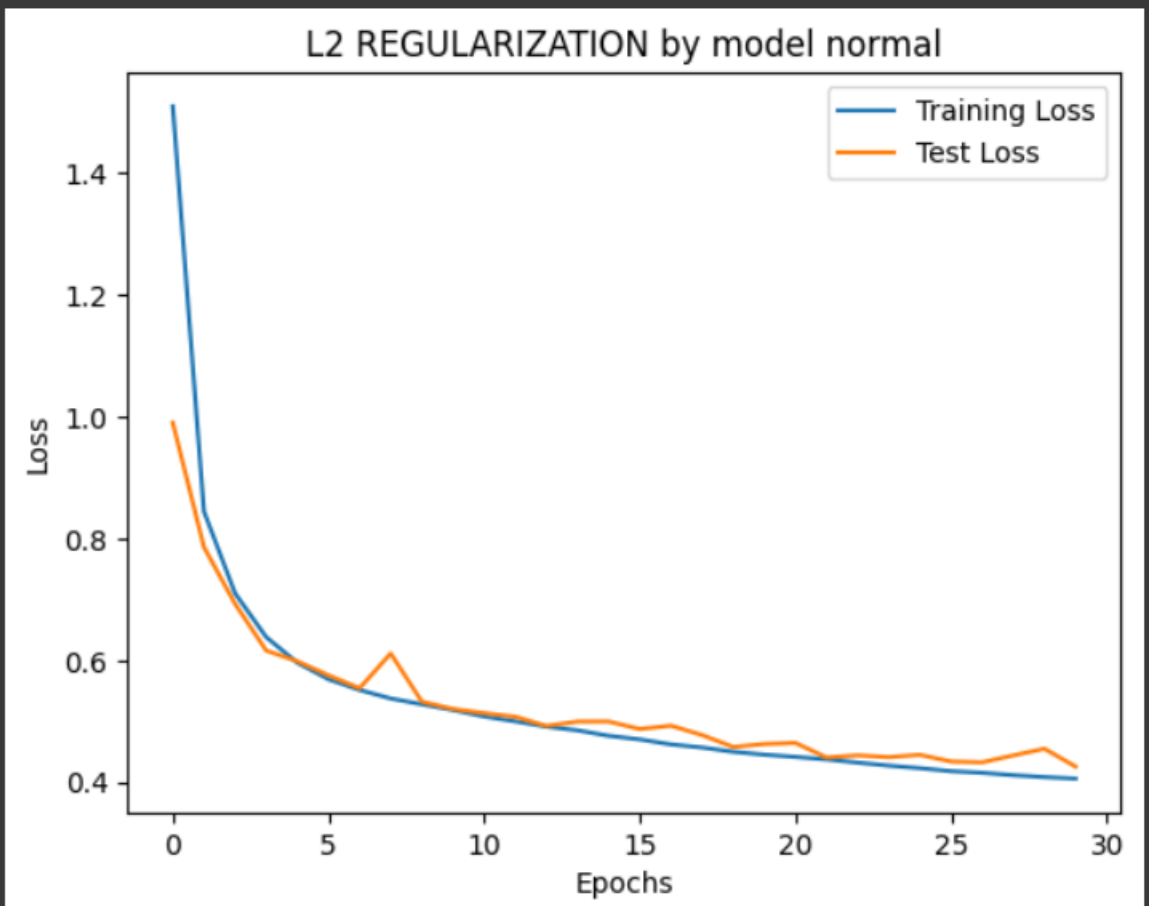
همانگونه که در دو نمونه ی پیاده سازی شده می بینید با این منظم سازی جلوی اورفیتینگ گرفته شدی و اختلاف و فاصله ا ی بین داده ای تست و اموزش مشاهده نمی شوند هرچند که موارد ترکیبی یا داده فزایی بیش تر روی مقدار خطا تاثیر گذاشتند

```
[36] Training loss: 0.3855249818199987
Test loss: 0.407172292470932
Training loss: 0.381324563040408
Test loss: 0.3948929011821747
Training loss: 0.3777497827148895
Test loss: 0.4022247791290283
```




test & train normal model for 30 epochs
Training Loss
Test Loss
LOSS
EPOCHS
1.6
1.4
1.2
1.0
0.8
0.6
0
5
10
15
20
25
30

```
Test loss: 0.45497965812683105
Training loss: 0.40604389636819044
Test loss: 0.4257202744483948
```



L2 based weight decay can also be implemented by setting a delta value for `weight_decay` in the optimizer.
weight_decay (*float*, *optional*) – weight decay (L2 penalty) (default: 0)

PyTorch (n.d.)

ث) شایان ذکر است که Dropout* در واقع چیزی بیش از منظم‌سازی انجام می‌دهد، این مدل را قوی‌تر می‌کند و به آن اجازه می‌دهد گره‌های مختلف را برای پیش‌بینی امتحان کند.

Dropout به حذف واحدها در یک شبکه عصبی اشاره دارد. با حذف یک واحد، به معنای حذف موقت آن از شبکه است. انتخاب واحدهایی که باید رها شوند تصادفی است. هر واحد با احتمال ثابت p مستقل از واحدهای دیگر حفظ می شود.

این روش به طور موثر مدل‌های کمی متفاوت با توپولوژی‌های نورون متفاوت در هر تکرار ایجاد می‌کند، بنابراین به نورون‌های مدل، فرصت کمتری برای هماهنگی در فرآیند به خاطر سپردن که در طول بیش‌برازش اتفاق می‌افتد، می‌دهد. بنابراین آن را در تعمیم بهتر و مقابله با موضوع بیش از حد مناسب می کند.

در مورد L1/L2، فقط با جریمه کردن وزنه های بالاتر، اضافه برازش را کاهش می دهد

خلاصه یکی از مقالات درباره ی ترکیب روش ها :

تکنیک های یادگیری عمیق به دلیل ساختار لایه ای پیچیده با مشکل بیش از حد برازش مواجه می شوند. برای غلبه بر این مشکل و بهبود مدل های طراحی شده از روش های منظم سازی استفاده می شود. در این مقاله از ترکیب روش های تنظیم L1، منظم سازی L2، Elastic Net-regularization و Dropout استفاده می کنیم. مدل عمیق طراحی شده با استفاده از ترکیب این روش ها با نرخ های متفاوتی در نظر گرفته شده است. مدل شبکه عمیق با استفاده از ترکیبی از این روش ها با نرخ های مختلف طراحی شده است. در نهایت، عملکرد همه روش‌های ترکیبی با مدل شبکه عصبی کانولوشن که از روش‌های منظم‌سازی استفاده نمی‌کند، مقایسه می‌شود. آزمایش ها با استفاده از مجموعه داده قیمت طلا در هر اونس و مدل شبیه سازی خطی انجام می شود. نتایج به‌دست‌آمده نشان می‌دهد که عملکرد مدل ترکیبی Dropout و منظم‌سازی شبکه الاستیک بهتر از مدل‌های دیگر است.

ترکیب روشهای منظم سازی با یکدیگر میتواند مزایا و معایبی داشته باشد. برخی از مزایای ترکیب روشهای منظم سازی عبارتند از:

- افزایش قدرت تعمیم پذیری مدل به دادههای جدید و کاهش خطای عمومی
- کاهش احتمال بیش برازش و وابستگی مدل به دادههای آموزشی
- ایجاد تنوع و تعادل بین روشهای مختلف منظم سازی و جبران کردن نقاط ضعف هر یک
- امکان انتخاب بهینهترین ترکیب و ضرایب منظم سازی با استفاده از روشهای اعتبارسنجی متقاطع

برخی از معایب ترکیب روشهای منظم سازی عبارتند از:

- افزایش پیچیدگی مدل و زمان آموزش

- احتمال کم برازش و کاهش قدرت یادگیری مدل در صورت اعمال منظم سازی بیش از حد

- ایجاد تضاد و تعارض بین روشهای مختلف منظم سازی و کاهش کارایی هر یک

- دشواری در تفسیر و تحلیل نتایج و اثرات روشهای منظم سازی بر روی مدل

بنابراین، ترکیب روشهای منظم سازی با یکدیگر میتواند خوب یا بد باشد، بسته به مسئله، مدل، دادهها و هدفی که داریم. برای انتخاب بهترین ترکیب و ضرایب منظم سازی، باید از روشهای اعتبارسنجی متقاطع و معیارهای ارزیابی مناسب استفاده کرد

Method	**Test Classification error %**
L2	1.62
L2 + L1 applied towards the end of training	1.60
L2 + KL-sparsity	1.55
Max-norm	1.35
Dropout + L2	1.25
Dropout + Max-norm	**1.05**

Table 9: Comparison of different regularization methods on MNIST.

ما دریافتیم که دراپ اوت همراه با منظم‌سازی حداکثر هنجار کمترین خطای تعمیم را می‌دهد.

پیاده سازی در شبکه های ما :

برای اینکه تست و ترین داده های ما زمان بسیار زیادی میبرد از این رو فقط برای این قسمت از ۳۰ ایپاک استفاده شده !

قسمت های قبلی با ۶۰ ایپاک هر کدام ۱ ساعت زمان برد :))

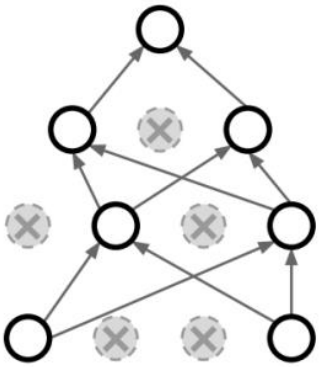
مانند این اسلاید دراپ اوت انجام دادیم و منظم سازی های دیگر را نیز در قسمت قبل پیاده سازی کردیم

Dropout

```
= 0.5 # probability of keeping a unit active. higher = less dropout
def train_step(X):
 # forward pass for example 3-layer neural network
 H1 = np.maximum(0, np.dot(W1, X) + b1)
 U1 = (np.random.rand(*H1.shape) < p) # first dropout mask.
 H1 *= U1 # drop!
 H2 = np.maximum(0, np.dot(W2, H1) + b2)
 U2 = (np.random.rand(*H2.shape) < p) # second dropout mask.
 H2 *= U2 # drop!
 out = np.dot(W3, H2) + b3

 # backward pass: compute gradients... (not shown)
 # perform parameter update... (not shown)

def predict(X):
 # ensembled forward pass
 H1 = np.maximum(0, np.dot(W1, X) + b1) * p # scale the activations
 H2 = np.maximum(0, np.dot(W2, H1) + b2) * p # scale the activations
 out = np.dot(W3, H2) + b3
```

drop in forward pas

scale at test time



ترکیب L2 و دیتافزایی

مدل توانسته خوب عمل بکند و اورفیت نداریم در شبکه اما موارد بعدی علکردی بهتر از این عملکرد این مورد دارند

```
Test loss: 0.444789856672287
Training loss: 0.41833098243866396
Test loss: 0.4451601207256317
Training loss: 0.4146643210925273
Test loss: 0.4237830340862274
Training loss: 0.41063203411633525
Test loss: 0.4320526719093323
Training loss: 0.408109394186087
 Test loss: 0.4196133017539978
```

ردار خروجی با اندازه ۱۰ تبدیل می کند. این لایه نماینده ۱۰ کلاس مختلف اعداد از ۰ تا ۹ است. این لایه نیز یک وزن و یک بایاس دارد که در طول آموزش بهینه می شوند.

- یک لایه LogSoftmax که یک تابع فعال سازی خطی است و بردار ورودی را به یک بردار احتمالاتی تبدیل می کند. این لایه مقدار لگاریتم احتمال هر کلاس را بر می گرداند. این لایه برای محاسبه خطا و به روز رسانی وزن ها و بایاس ها با استفاده از الگوریتم بهینه سازی مورد نیاز است.

- این کد از بسته torchvision.transforms برای آماده سازی و بارگذاری داده ها استفاده می کند که شامل چندین تابع تغییر و افزایش داده های تصویری است. این کد از توابع زیر استفاده می کند:

- transforms.Compose: یک تابع که چندین تبدیل را در یک لیست قرار می دهد و یک تبدیل جدید ایجاد می کند که تمام تبدیل های لیست را به ترتیب اعمال می کند.

- transforms.RandomRotation: یک تابع که تصویر را با یک زاویه تصادفی در بازه داده شده چرخانده می کند.

- transforms.RandomHorizontalFlip: یک تابع که تصویر را با احتمال ۰,۵ افقی معکوس می کند.

- transforms.ToTensor: یک تابع که تصویر را از یک شیء PIL یا numpy array به یک تنسور pytorch تبدیل می کند.

این کد دو تبدیل مختلف برای داده های آموزش و آزمون ایجاد می کند. تبدیل آموزش شامل چرخش و معکوسی افقی تصادفی است که برای افزایش تنوع داده ها و جلوگیری از بیش برازش مفید است. تبدیل آزمون فقط شامل تبدیل تصویر به تنسور است که برای سازگاری با مدل لازم است.

این کد همچنین از کلاس FashionMNIST برای بارگذاری مجموعه داده استفاده می کند. این کلاس یک زیر کلاس از کلاس Dataset است که یک رابط برای دسترسی به داده ها را فراهم می کند. این کلاس چند پارامتر را می پذیرد:

- root: مسیر پوشه ای که داده ها در آن ذخیره می شوند.

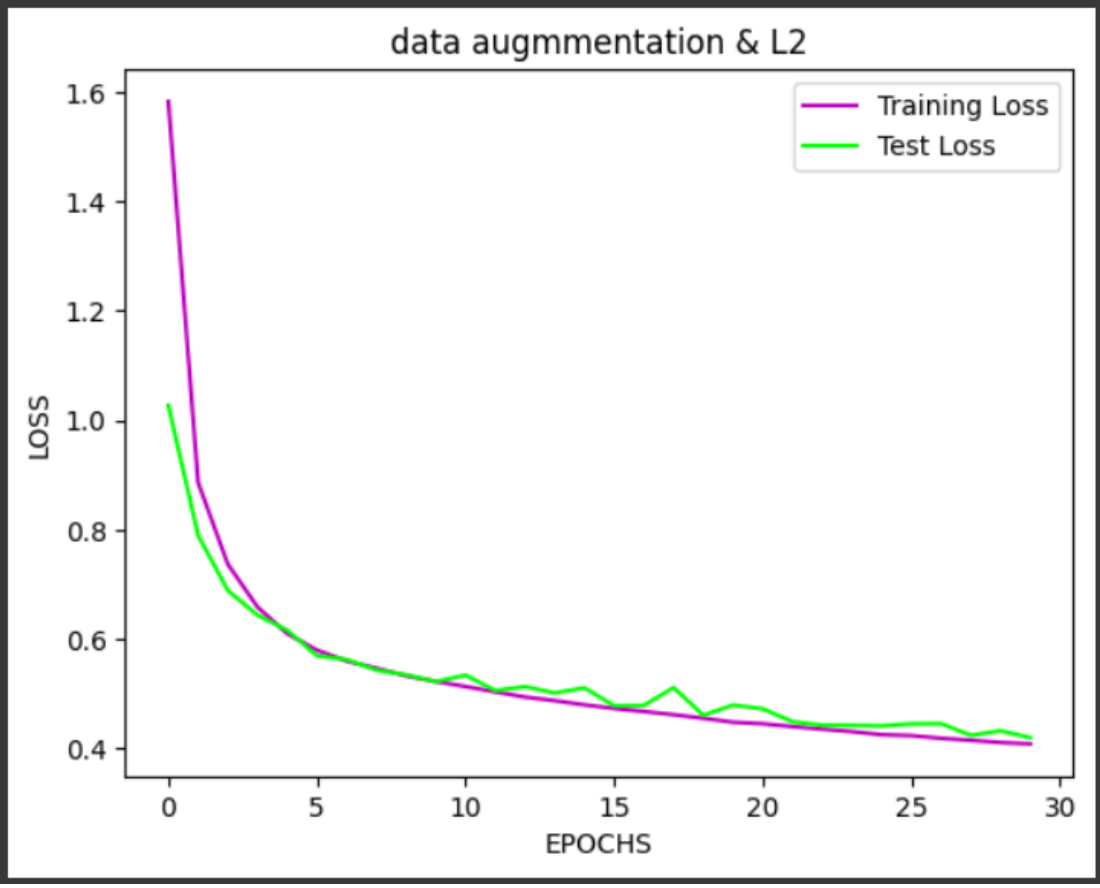
- train: یک مقدار بولی که مشخص می کند که داده های آموزش یا آزمون بارگذاری شوند.

- download: یک مقدار بولی که مشخص می کند که آیا داده ها از اینترنت دانلود شوند یا خیر.

- transform: یک تابع که بر روی تصاویر اعمال می شود.

برای تقسیم داده ها به دسته های کوچکتر و قابل مدیریت تر استفاده می کند. این DataLoader این کد همچنین از کلاس کلاس یک ابزار برای بارگذاری داده ها به صورت موازی و کارآمد است. این کلاس چند پارام

```
Training loss: 0.4146643210925273
Test loss: 0.4237830340862274
Training loss: 0.41063203411633525
Test loss: 0.4320526719093323
Training loss: 0.408109394186087
Test loss: 0.4196133017539978
```

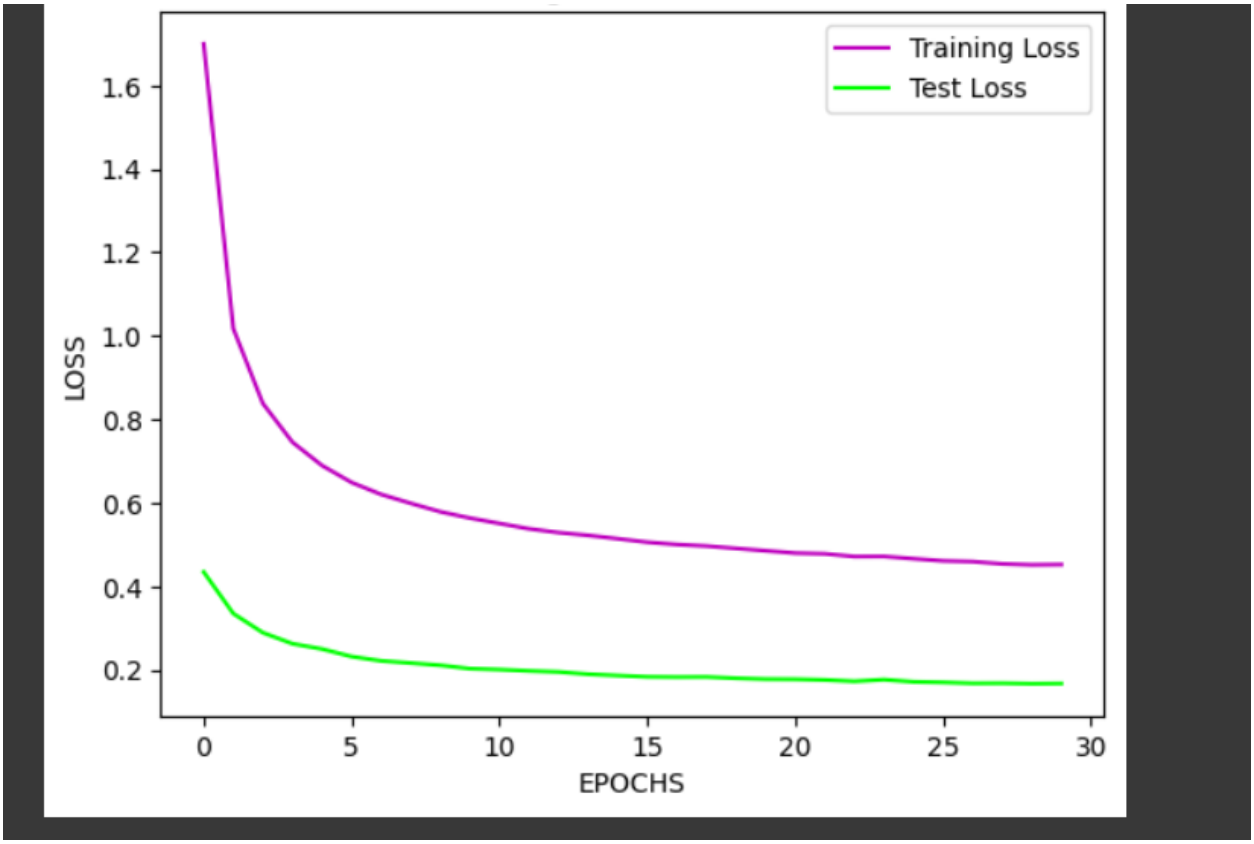


ترکیب دراپ اوت و دیتافزایی

به خوبی جلوی اورفیتیگ را گرفته داده ها به درستی فیت شده اند در کد هم در قسمت اموزش با احتمال 0.4. نورن را غیر فعال می کنیم و بعد ساختا ری که برای داده فزایی استفاده شده بود را پیاده سازی می کنیم در مرحله تست هم در یک ۴ دهم ضرب می کنیم

```
Test loss: 0.1774638444185257
Training loss: 0.4781224118080983
Test loss: 0.17586316168308258
Training loss: 0.4715293202318871
Test loss: 0.17254355549812317
Training loss: 0.4717364111847715
Test loss: 0.1765557825565338
```

```
Training loss: 0.4662633352378792
Test loss: 0.1712537705898285
Training loss: 0.46108906198221483
Test loss: 0.16995449364185333
Training loss: 0.45937663018068015
Test loss: 0.16770564019680023
Training loss: 0.45385030847686186
Test loss: 0.16790485382080078
Training loss: 0.4516276761508191
Test loss: 0.16679666936397552
Training loss: 0.4522634225645299
Test loss: 0.16725030541419983
```



نحوه ی عملکرد : بسیار عالی

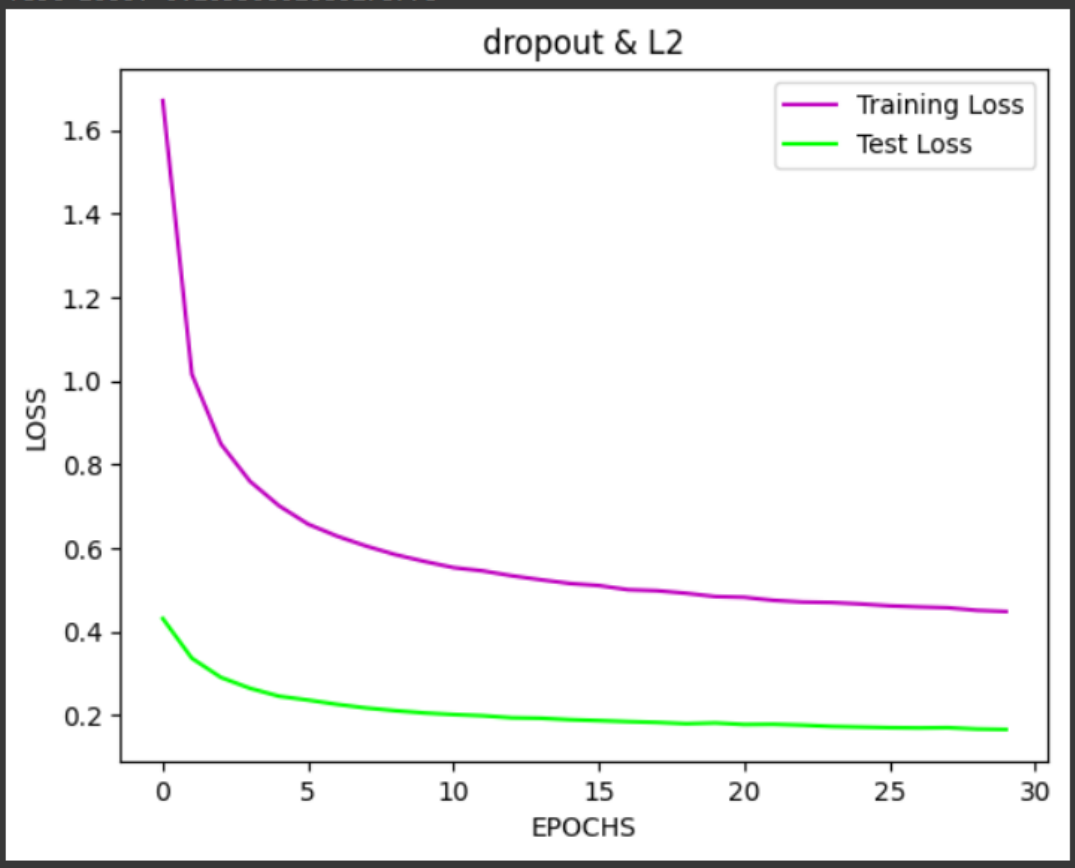
لاس تست به ۰,۱۶ رسیده که بسیار عدد خوب و قابل توجهی است و ترکیب این دو مورد توانسته بسیار خوب بایکدیگر کار کند

ترکیب l2 و دراپ اوت

توضیح کد :

دراپ اورت را با اختمال ۴ دهم پیاده سازی کردیم و پارامتر کاهششی وزن را با مقدار مشخص تعمیم یافته برای L2 استفاده می کنیم

```
Training loss: 0.456529599834861
Test loss: 0.16953244805335999
Training loss: 0.45027723646303736
Test loss: 0.1662856638431549
Training loss: 0.44752998683434814
Test loss: 0.16530601680278778
```



ترکیب های با دراپ ات به خوبی توانسته که در قسمت تست عمل کند و مدل به خوبی تعمیم یافته شده و در برابر جواب هایی هم که ندیده است به خوبی عمل می کند

نحوه ی عملکرد : بسیار عالی

ترکیب این موارد برای من بهتر از استفاده ی جداگانه هر کدام از منظم سازی ها کار کرد و بهترین مدل ها را با کمترین خطای پیش بینی و دقت زیاد در شرایط یکسان با بقیه ی مدل ها به ما داد

به ترتیب در این مدل ها بهترین عملکرد برای ترکیب l2 و دراپ اوت

با کم ترین مقدار لاس در تست ۰،۱۶۵۳ , و منظم سازی مناسب برای دیتا ها و پیش بینی بسیارر مناسب حتی کم تر از مقدار داده های اموزشی

در رتبه بعدی دراپ اوت و دیتا فزایی

```
Test loss: 0.16725030541419983
```

و در رتبه ی بعد

ترکیب L2 و دیتافزایی

```
Test loss: 0.4196133017539978
```

ترکیب این موارد با یکدیگر نتایج مفید و مثمر ثمری را به ما داد که از اعمال تنهایی ان ها مقدار لاس را بیش تر کاهش البته این ترکیب ها برای دیتاست های مختلف و با توجه به مدا متفاوت می باشد

https://stats.stackexchange.com/questions/241001/deep-learning-use-l2-and-dropout-regularization-simultaneously

https://ieeexplore.ieee.org/document/9986657

https://pytorch.org/vision/master/generated/torchvision.transforms.v2.RandomChoice.html#torchvision.transforms.v2.RandomChoice