پاسخ تمرین دوم **مهدیه نادری: ۹۸۵۲۲۰۷۶**

سوال اول)

۱.الف بیش برازش و کم برازش در شبکه‌های عصبی:

بیش برازش:

بیش برازش (Overfitting) یکی از مشکلات رایج در آموزش شبکه‌های عصبی است که وقتی مدل به طور زیادی به داده‌های آموزشی خود متکی شده و نتواسته از داده‌های جدید به خوبی عمل کند، اتفاق می‌افتد. به عبارت دیگر، مدل در فرآیند آموزش به نقاط و جزئیات کوچکی از داده‌های آموزشی پاسخ می‌دهد و این باعث می‌شود که برای داده‌های جدید و نادیده نتواند به خوبی عمل کند.

دلایل بیش برازش ممکن است شامل موارد زیر باشد:

1. **مدل پیچیده‌تر از اندازه داده‌ها:** اگر مدل بسیار پیچیده باشد و تعداد داده‌های آموزشی کم باشد، ممکن است مدل به طور غیرمناسب به نقاط داده‌های آموزشی بپیوندد.
2. **تعداد داده‌های آموزشی کم :** اگر تعداد داده‌های آموزشی کم باشد نسبت به پیچیدگی مدل، ممکن است مدل به داده‌های آموزشی خود بیش از حد منحصر شود.
3. **عدم تنظیم میزان(Regularization)** : تنظیم میزان یک روش موثر برای کنترل بیش برازش است. در تنظیم میزان، ایجاد محدودیت‌هایی بر روی وزن‌ها و یا افزودن جملات جریمه کننده به تابع هدف کمک می‌کند تا جلوی بیش برازش گرفته شود.
4. **نویز در داده‌ها:** وجود نویز در داده‌های آموزشی ممکن است باعث شود مدل به نقاط داده‌های آموزشی به نحوی واکنش نشان دهد که در واقع نویز را یاد بگیرد و این باعث بیش برازش می‌شود.
5. **انتخاب اشتباه ویژگی‌ها (Feature Selection)** : اگر ویژگی‌های انتخاب شده برای آموزش مدل به طور ناکارآمدی توانایی تعمیم‌پذیری را نشان دهند، این می‌تواند منجر به بیش برازش شود.

برای جلوگیری از بیش برازش، می‌توان از روش‌هایی مانند تنظیم میزان، استفاده از داده‌های بیشتر، استفاده از مدل‌های ساده‌تر، و انتخاب صحیح ویژگی‌ها استفاده کرد. همچنین، استفاده از روش‌های ارزیابی مانند ارزیابی متقابل (Cross-Validation) نیز می‌تواند کمک کننده باشد تا بیش برازش در مدل‌های عصبی کاهش یابد.

کم برازش:

کم برازش (Underfitting) در شبکه‌های عصبی به وقوع می‌پیوندد وقتی مدل به طور کلی نمی‌تواند الگوهای مهم در داده‌های آموزشی را یاد بگیرد. این مشکل ممکن است به دلیل سادگی بیش از حد مدل یا کم بودن تعداد لایه‌ها و نرون‌ها در شبکه عصبی باشد. علاوه بر این، دلایل دیگری که ممکن است به کم برازش منجر شوند عبارتند از:

1. **تعداد کم داده‌های آموزشی :**اگر تعداد داده‌های آموزشی کم باشد، مدل اطلاعات کافی برای یادگیری الگوهای دقیق نخواهد داشت و ممکن است به کم برازش برخورد کند.
2. **انتخاب ویژگی‌های ناکافی :**اگر ویژگی‌های انتخاب شده برای آموزش مدل، الگوهای مهم در داده‌ها را نمایش ندهند، مدل نمی‌تواند به درستی یادگیری کند.
3. **تنظیم میزان بیش از حد :(Over-regularization)** استفاده از روش‌های تنظیم میزان برای جلوگیری از بیش برازش ممکن است به کم برازش منجر شود اگر بیش از حد محدودیت‌هایی روی مدل اعمال شود.
4. **نقص در پیش‌پردازش داده‌ها :**اگر داده‌های ورودی به مدل بدون پیش‌پردازش یا استانداردسازی نباشند، ممکن است که مدل به درستی عمل نکند.

برای رفع مشکل کم برازش، می‌توان از راهکارهای زیر استفاده کرد:

1. **افزایش تعداد داده‌های آموزشی :**اگر امکان افزایش تعداد داده‌های آموزشی وجود دارد، می‌توانید از این راه برای کمک به مدل در یادگیری الگوها استفاده کنید.
2. **انتخاب ویژگی‌های مناسب :**اطمینان حاصل کنید که ویژگی‌های انتخاب شده برای آموزش مدل، الگوهای مهم و معنادار در داده‌ها را نمایش می‌دهند.
3. **ساده‌تر کردن مدل :**اگر مدل بسیار پیچیده است، ممکن است ساده‌ترش کنید تا بهترین تطابق را با داده‌ها داشته باشد.
4. **بهبود پیش‌پردازش داده‌ها :**داده‌های ورودی را ممکن است نیاز باشد پیش‌پردازش کنید، از جمله استفاده از مقیاس‌دهی، نرمال‌سازی و یا حتی از روش‌های پیچیده‌تر مانند افزایش ابعاد ویژگی‌ها.

۱.ب تشخیص بیش برازش در مدل از قبل آموزش دیده:

چندین روش برای تشخیص اینکه یک مدل اورفیت کرده است یا خیر ذکر شده است:

۱ .عملکرد در مجموعه آموزش و اعتبارسنجی:

- عملکرد مدل را در مجموعه آموزش و یک مجموعه اعتبارسنجی مجزا مقایسه کنید. اگر عملکرد مدل بر روی مجموعه آموزش به طور قابل توجهی بهتر از عملکرد بر روی مجموعه اعتبارسنجی باشد، این ممکن است نشان‌دهنده وجود اورفیت باشد.

۲ .نمودارهای آموزش:

- نمودارهای آموزش را رسم کنید که عملکرد آموزش و اعتبارسنجی را نشان دهد. اگر عملکرد آموزش به طور مداوم بهبود یابد در حالی که عملکرد اعتبارسنجی به سطح برخوردهای یا بهبود نشان ندهد، این نشان‌دهنده اورفیتینگ است.

۳ .استفاده از اعتبارسنجی متقابل:

- اگر به یک مجموعه داده برچسب‌گذاری شده دسترسی دارید، می‌توانید اعتبارسنجی متقابل انجام دهید. داده را به بخش‌های آموزش و اعتبارسنجی تقسیم کنید، مدل را روی یک بخش آموزش بگذارید و روی بخش دیگر ارزیابی کنید. این روش را چندین بار تکرار کنید. اگر تغییرات قابل توجهی در عملکرد به وجود آید، ممکن است نشان‌دهنده اورفیتینگ باشد.

۴ .تکنیک‌های تنظیم میزان:

- اگر در هنگام آموزش از تکنیک‌های تنظیم میزان مثل dropout یا تنظیم میزان وزن استفاده کردید، تأثیر آنها را بررسی کنید. اگر تنظیم میزان مؤثر باشد، این به کاهش اورفیتینگ کمک خواهد کرد.

۵ .مقایسه تابع هزینه و دقت در داده‌های آموزش و اعتبارسنجی:

- تغییرات در مقادیر تابع هزینه (یا خطا) و دقت را در طول آموزش مشاهده کنید. اگر تابع هزینه بر روی داده‌های آموزش به سرعت کاهش یابد اما بر روی داده‌های اعتبارسنجی افزایش یابد، ممکن است نشان‌دهنده اورفیتینگ باشد.

۶ .ارزیابی بر روی مجموعه آزمون:

- اگر امکان دارد، مدل را بر روی یک مجموعه آزمون کاملاً ناشناخته ارزیابی کنید. اگر کاهش عملکرد نسبت به مجموعه اعتبارسنجی به وجود آید، این ممکن است نشان‌دهنده اورفیتینگ باشد.

۷ .استفاده از مدل‌های انسمبل:

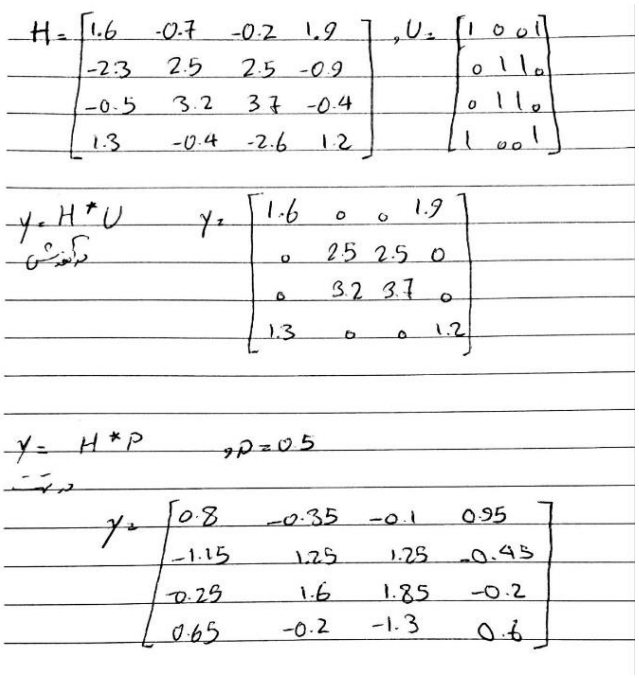
- یک انسمبل از مدل‌ها (مانند بگینگ) بسازید و عملکرد آن را ارزیابی کنید. انسمبل می‌تواند به کاهش اورفیتینگ کمک کند و اگر عملکرد انسمبل بهتر از مدل‌های فردی باشد، نشان‌دهنده وجود اورفیتینگ در مدل‌های فردی است.

۸ .استفاده از مدل ساده‌تر:

- یک مدل ساده‌تر با تعداد پارامترهای کمتر آموزش دهید و عملکرد آن را با مدل پیچیده مقایسه کنید

۱.پ

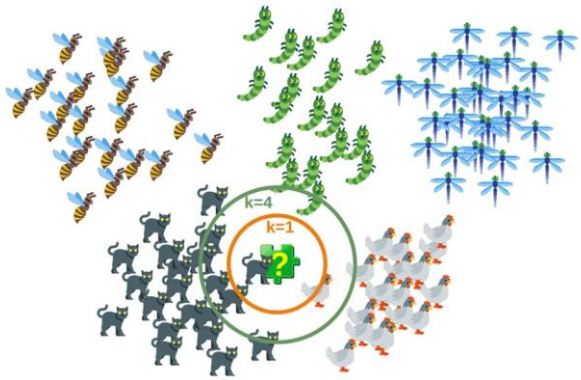
باتوجه به ماتریس U فهمیدیم P=0.5 است.



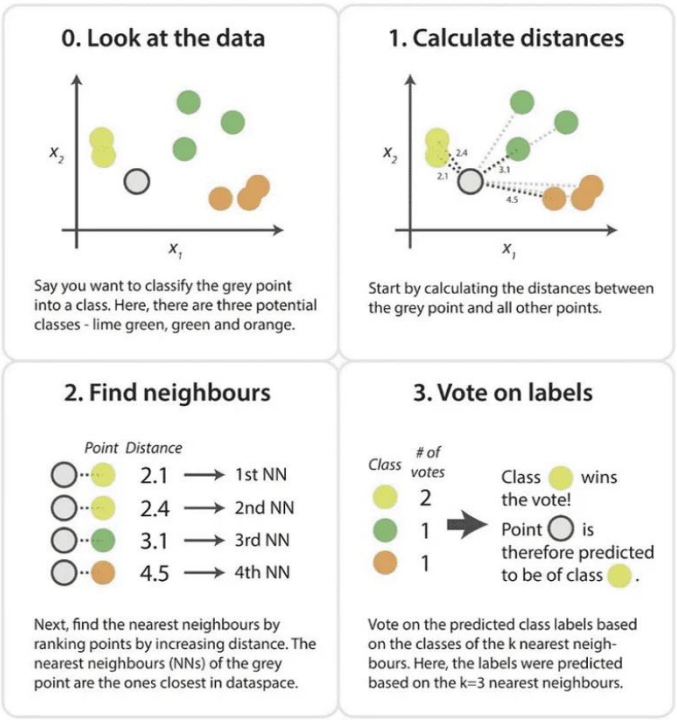
سوال دوم:

۱.الف تاثیر تغییر K روی بایاس و واریانس در الگوریتم نزدیکترین همسایگی:

الگوریتم نزدیکترین همسایه:



K-nearest neighbors (KNN) یک نوع الگوریتم یادگیری نظارت شده است که برای رگرسیون و طبقه بندی استفاده می شود. KNN سعی می کند با محاسبه فاصله بین داده های آزمون و تمام نقاط آموزشی کلاس صحیح را برای داده های آزمون پیش بینی کند. سپس K تعداد نقاط را که نزدیک به داده های تست است انتخاب کنید. الگوریتم KNN احتمال داده‌های آزمون متعلق به کلاس‌های داده‌های آموزشی «K» را محاسبه می‌کند و کلاس دارای بالاترین احتمال انتخاب می‌شود. در مسائل رگرسیون، مقدار میانگین امتیازهای آموزشی انتخاب شده «K» است.



فرض کنید، تصویری از موجودی داریم که شبیه گربه و سگ است، اما می‌خواهیم بدانیم که گربه است یا سگ. بنابراین برای این شناسایی، می‌توانیم از الگوریتم KNN استفاده کنیم، زیرا بر روی معیار شباهت کار می‌کند. مدل KNN ما ویژگی‌های مشابه مجموعه داده‌های جدید را با تصاویر گربه‌ها و سگ‌ها پیدا می‌کند و بر اساس مشابه‌ترین ویژگی‌ها، آن را در دسته‌بندی گربه یا سگ قرار می‌دهد.

K-NN چگونه کار می کند؟

کار K-NN را می توان بر اساس الگوریتم زیر توضیح داد:

مرحله 1: عدد K همسایه ها را انتخاب کنید

مرحله 2: فاصله اقلیدسی K تعداد همسایگان را محاسبه کنید

مرحله 3: نزدیکترین همسایگان K را بر اساس فاصله اقلیدسی محاسبه شده بگیرید.

مرحله 4: از میان این k همسایه ها، تعداد نقاط داده در هر دسته را بشمارید.

مرحله 5: نقاط داده جدید را به دسته ای اختصاص دهید که تعداد همسایگان برای آن حداکثر است.

مرحله 6: مدل ما آماده است.

K میتواند مقادیر متفاوتی داشته باشد و به مسئله بستگی دارد.

1. **تأثیر بر بایاس:**
   - زیاد کردن مقدار K معمولاً منجر به افزایش بایاس مدل می‌شود. زیرا با افزایش تعداد همسایگان نزدیک، مدل به میانگین‌گیری بیشتر از اطراف هدف می‌پردازد. این ممکن است منجر به از دست رفتن جزئیات دقیق‌تر در داده‌ها شود و بایاس مدل را افزایش دهد.
2. **تأثیر بر واریانس:**
   - افزایش مقدار K ممکن است باعث کاهش واریانس مدل شود. زیرا با افزایش تعداد همسایگان، مدل به ساختار کلی تری از داده‌ها پی می‌برد و حساسیت به نویزها و نواحی محلی کمتر می‌شود. این ممکن است باعث افزایش پایداری و کاهش تغییرات ناخواسته در پیش‌بینی‌ها شود.

بنابراین، با افزایش مقدار:K

- بایاس افزایش می‌یابد.
- واریانس کاهش می‌یابد.

به هنگام انتخاب مقدارK ، نیاز به یک توازن مناسب بین بایاس و واریانس وجود دارد. اگر K خیلی کوچک باشد، ممکن است مدل به نواحی محلی داده‌ها حساس شود (واریانس بالا) و اگر K خیلی بزرگ باشد، ممکن است اطلاعات دقیق‌تر در داده‌ها از دست برود (بایاس بالا). انتخاب مناسب برای K بستگی به مسئله خاص و داده‌های مورد استفاده دارد.

۲.ب درستی و نادرستی گزاره‌ها:

- **استفاده از منظم سازی ممکن است باعث تضعیف مدل شود: درست**

استفاده از منظم‌سازی (Regularization) ممکن است در برخی مواقع باعث تضعیف مدل شود. منظم‌سازی معمولاً با هدف کاهش بیش از حد واریانس (Overfitting) مدل به کار می‌رود، اما در برخی شرایط ممکن است تأثیر منفی بر عملکرد مدل داشته باشد. دلایلی که موجب تضعیف مدل به وسیله منظم‌سازی می‌شوند عبارتند از:

1. **منظم‌سازی اضافی:(Over-regularization)**

   اگر میزان منظم‌سازی بیش از حد باشد، ممکن است باعث از دست رفتن اطلاعات مفید در داده‌ها شود و مدل به شدت ساده شده و ضعیف شود.
2. **تنظیم نادرست پارامترهای منظم‌سازی:**

   انتخاب نادرست پارامترهای منظم‌سازی، مانند میزان تنظیم (lambda) در منظم‌سازی L1 یاL2 ، ممکن است باعث تأثیر منفی بر عملکرد مدل شود. انتخاب مناسب این پارامترها امری حیاتی است.
3. **ناکافی بررسی پارامترهای مدل:**

   اگر پارامترهای مدل به درستی تنظیم نشوند و به اندازه کافی بررسی نشوند، ممکن است منظم‌سازی تضعیف کننده شود. بهتر است این پارامترها با استفاده از اعتبارسنجی (validation) به درستی انتخاب شوند.
4. **منظم‌سازی در مسائل کم‌ابعاد:**

   در مسائلی که داده‌ها به تعداد کمی از نمونه‌ها یا ویژگی‌ها محدود است، استفاده از منظم‌سازی ممکن است باعث از دست رفتن اطلاعات مهم شود و مدل را ضعیف کند.

در کل، برای استفاده مؤثر از منظم‌سازی، نیاز است تا با دقت پارامترهای آن تنظیم شوند و به ویژه در نظر گرفته شود که از اندازه‌گیری مناسبی از منظم‌سازی برای مسئله مورد نظر استفاده شود. همچنین، توجه به نیازمندی‌های خاص مسئله و ماهیت داده‌ها نیز اهمیت دارد.

- **اضافه کردن تعداد زیاد فیچرهای جدید، باعث جلوگیری از بیش برازش میشود: غلط**

پاسخ تمرین دوم **مهدیه نادری: ۹۸۵۲۲۰۷۶**

افزودن تعداد زیادی از ویژگی‌ها (فیچرها) به مدل، به طور عام ممکن است باعث جلوگیری از بیش برازش نشود، بلکه ممکن است باعث افزایش احتمال بیش برازش شود. اضافه کردن فیچرها می‌تواند به تعداد پارامترهای مدل افزوده شود که ممکن است به پیچیدگی افزوده شود و در نتیجه، احتمال ایجاد بیش برازش افزایش یابد.
به عبارت دیگر، اضافه کردن ویژگی‌ها به مدل باید با اهمیت به توازن مناسب میان پیچیدگی مدل و تعداد داده‌ها صورت گیرد. موارد زیر نشان‌دهنده تأثیر افزودن ویژگی‌ها بر بیش برازش و عملکرد مدل هستند:

1. **تعداد داده‌ها:**
   اگر تعداد داده‌ها نسبت به تعداد ویژگی‌ها کم باشد، افزودن ویژگی‌ها ممکن است باعث بیش برازش شود. به عبارت دیگر، افزایش پیچیدگی مدل در صورت کمبود داده ممکن است منجر به بیش برازش شود.
2. **اهمیت و انتخاب ویژگی‌ها:**
   اگر ویژگی‌ها با دقت انتخاب نشوند یا اگر بی‌اهمیت باشند، افزودن زیادی از ویژگی‌ها ممکن است به افزایش پیچیدگی مدل و بیش برازش منجر شود. اهمیت و انتخاب ویژگی‌ها باید با دقت انجام شود.
3. **توان مفرط ویژگی‌ها:**
   اگر ویژگی‌ها توانمندی‌ها یا اطلاعات مشابه را ارائه دهند، اضافه کردن زیادی از آنها ممکن است به افزایش پیچیدگی و بیش برازش منجر شود.
4. **استفاده از روش‌های کاهش بعد:**
   در مواردی که تعداد ویژگی‌ها زیاد است، می‌توان از روش‌های کاهش بعد مانند تحلیل مؤلفه اصلی (PCA) یا انتخاب ویژگی‌های مهم استفاده کرد تا تعداد ویژگی‌ها را بهینه کنیم و از افزایش بیش برازش جلوگیری کنیم.

از این رو، مهم است که در افزودن ویژگی‌ها به مدل، دقت و کیفیت این ویژگی‌ها و تأثیر آنها بر عملکرد مدل مورد بررسی قرار گیرد و با دقت و تعقیب انجام شود.

- **با زیاد کردن ضریب منظم سازی احتمال بیش برازش بیشتر میشود:درست**

بله، با زیاد کردن ضریب منظم‌سازی) معمولاً از طریق پارامترهای مثل ضریب تنظیم در تکنیک‌های مانند L1 یا(L2 regularization ، احتمال بیش‌برازش (Overfitting) افزایش می‌یابد. منظم‌سازی معمولاً برای کنترل بیش‌برازش به کار می‌رود و با افزایش ضریب منظم‌سازی، مدل به یک شکل ساده‌تر و کمتر پیچیده تبدیل می‌شود.
ضریب منظم‌سازی عمدتاً به یک توازن مناسب بین بایاس و واریانس مدل کمک می‌کند. افزایش ضریب منظم‌سازی به این معناست که مدل به میزان بیشتری مجاز به استفاده از پارامترهای پیچیده تر می‌شود، و این می‌تواند به جلوگیری از پیچیدگی اضافی در مدل و از بین بردن برخی پارامترهای اضافی کمک کند.
با افزایش ضریب منظم‌سازی:

۱. **پارامترهای مدل کاهش می‌یابند:**
   مقادیر پارامترهای مدل به میزان بیشتری تنظیم می‌شوند و تعداد آنها کاهش می‌یابد.
۲. **پیچیدگی مدل کاهش می‌یابد:**
   مدل به سادگی‌تر می‌شود و پیچیدگی آن کاهش می‌یابد.
۳. **احتمال بیش‌برازش کاهش می‌یابد:**
   با کاهش تعداد پارامترها و ساده‌تر شدن مدل، احتمال بیش‌برازش کاهش می‌یابد.

با این حال، نباید ضریب منظم‌سازی را به شدت افزایش داد، زیرا این ممکن است منجر به بیش‌بهینگی (Underfitting) شود، به طوری که مدل به اندازه کافی پیچیده نباشد تا داده‌ها را به درستی یاد بگیرد. انتخاب مناسب ضریب منظم‌سازی نیازمند آزمون و خطا و استفاده از اعتبارسنجی (validation) است تا بهترین توازن بین دقت و اجتناب از بیش‌برازش را بیابیم.

۲.پ تشخیص منظم سازی L1 و L2 با توجه مقادیر وزن ها:

تفاوت های L1 و L2:

1. اثر بر وزن‌ها:
   - L2:
     - تمایل دارد وزن‌ها را به صورت پیوسته کم کند. این به این معناست که تمایل به کاهش تمام وزن‌ها به نسبت یکدیگر دارد، اما به صورت یکنواخت.
   - L1:
     - تمایل دارد وزن‌ها را به صورت گسسته (باینری) کند. این به این معناست که تمایل به تنظیم برخی از وزن‌ها به صفر دارد.
2. پایداری نسبت به داده‌های نویزی:
   - L2:
     - معمولاً مقاوم‌تر در برابر داده‌های نویزی است و ممکن است تاثیرات یک نقطه داده نویزی را به صورت یکنواخت‌تر جلوگیری کند.
   - L1:
     - اگرچه تاثیرات داده‌های نویزی را هم می‌تواند کاهش دهد، اما ممکن است به دلیل محاسبه مقدار مطلق وزن‌ها، به صورت پراکنده تر به داده‌های نویزی حساس باشد.

Wexp1 = [0.26, 0.25, 0.25, 0.25]:

در اینجا از منظم سازی L2 استفاده شده است چون وزنها یکنواخت هستند و فقط ۰,۰۱ اختلاف دارند. و وزن پاسخ‌های غلط ۰ نشده است و صرفا کمتر از ویژگی اصلی است تاثیر کمتری در خروجی بگذارند.

W exp2 = [1, 0, 0, 0]:

در اینجا از منظم سازی L1 استفاده شده است چون وزن اولین فیچر ۱ و بقیه ۰ شده اند. کاهش وزن در L1 به صورت باینری است و به فیچرهای غیر مهم وزن ۰ داده میشود.

Wexp3 = [13.3, 23.5, 53.2, 5.1]:

در این جا، مقادیر همه وزن‌ها بزگ است پس از L2 استفاده نشده است، از طرفی وزن هیچ فیچری صفر نیست پس L1 هم نیست. هیچ کدام☺

Wexp4 = [0.5, 1.2, 8.5, 0]:

در اینجا نیز ازمنظم سازی L1 استفاده شده است چون مقدار یکی از فیچرها صفر شده و این ویژگی L1 است از طرفی وزن های غیر صفر رنج متفاوتی دارند و یکنواخت نیستند پس L2 نیست چون آن مقادیر برای فیچر غیر اصلی را یکنواخت کم میکند.

پاسخ تمرین دوم **مهدیه نادری: ۹۸۵۲۲۰۷۶**

سوال سوم:

۳.الف تقطیر دانش:

تقطیر دانش روشی ساده برای بهبود عملکرد مدل‌های یادگیری عمیق روی دستگاه‌های موبایل است. در این فرآیند یک شبکه‌ی بزرگ و پیچیده یا یک مدل گروهی آموزش می‌دهیم که می‌تواند ویژگی‌های مهم داده‌ها را استخراج کرده و پیش‌بینی‌های خوبی انجام دهد. سپس به کمک این مدل سنگین، شبکه‌ای کوچک را آموزش می‌دهیم. این شبکه‌ی کوچک قادر خواهد بود نتایجی در حد قابل مقایسه (با مدل سنگین) ارائه دهد و در برخی موارد هم می‌تواند همان نتایج شبکه‌ی سنگین را تولید کند.

برای مثال GoogLeNet شبکه‌ای بسیار سنگین، یعنی عمیق و پیچیده است. عمقش به آن توان استخراج ویژگی‌های پیچیده را می‌دهد و پیچیدگیش به حفظ دقت کمک می‌کند. این مدل آنقدر سنگین است که برای پیاده‌سازی به حجم زیادی حافظه، یک GPU قدرتمند و محاسبات پیچیده نیاز خواهد داشت. به همین دلیل باید بتوانیم دانش این مدل را به یک مدل بسیار کوچک‌تر منتقل کنیم که قابلیت استفاده در یک دستگاه موبایل را دارد.

مدل‌های سنگین یاد می‌گیرند تا بین تعداد زیادی از دسته ها تمایز قائل شوند. هدف اصلی در آموزش , به حداکثر رساندن متوسط احتمال لگاریتمی پاسخ درست است و بدین منظور, مدل به هر دسته یک احتمال اختصاص می‌دهد (احتمال برخی طبقات از بعضی دیگر کوچک‌تر است). مقادیر نسبی احتمالات مربوط به پاسخ‌های اشتباه اطلاعات زیادی در مورد نحوه‌ی تعمیم‌پذیری این مدل پیچیده در اختیار ما می-گذارند. مثلاً تصویر یک ماشین به احتمال کمی به عنوان کامیون شناخته خواهد شد، اما احتمال وقوع همین اشتباه از احتمال تشخیص یک گربه (به جای ماشین) خیلی بیشتر است.

توجه داشته باشید که انتخاب تابع هدف باید به نحوی باشد که به خوبی به داده‌های جدید تعمیم داده شود. پس زمان انتخاب تابع هدف مناسب باید به خاطر داشته باشیم قرار نیست این تابع روی داده‌های آموزشی عملکردی بهینه داشته باشد.

از آن‌جایی که این عملیات برای دستگاه‌های موبایل بسیار سنگین است، باید دانش این مدل‌های سنگین را به یک مدل کوچک‌تر انتقال دهیم که به آسانی روی دستگاه‌های موبایل به کار برده شود. بدین منظور می‌توانیم مدل سنگین را شبکه‌ی معلم و مدل کوچک را شبکه‌ی دانش در نظر بگیریم.

می‌توانید شبکه‌ی بزرگ و پیچیده را به شبکه‌ای بسیار کوچک‌تر تقطیر کنید؛ این شبکه‌ی کوچک‌تر تابع اصلی را که توسط شبکه‌ی عمیق آموخته شده، به خوبی برآورد می‌کند.

یک نکته این‌جا وجود دارد و آن این است که مدل تقطیرشده (دانش‌آموز) به نحوی آموزش دیده که به جای آموزش مستقیم روی داده‌های خام، خروجی شبکه‌ی بزرگ‌تر (معلم) را تقلید کند؛ شاید به این خاطر که شبکه‌ی عمیق‌تر انتزاعات سلسله مراتبی ویژگی‌ها را می‌آموزد.

تقطیر دانش (Knowledge Distillation) یک روش در حوزه یادگیری عمیق (Deep Learning) است که با استفاده از یک مدل پیشین (معمولاً یک مدل پیچیده و بزرگ) به عنوان معلم(teacher) ، یک مدل کوچکتر و سبکتر به عنوان دانش‌جو (student) را آموزش می‌دهد. هدف از این روش انتقال دانش است که تا حد ممکن عملکرد مدل کوچکتر به عنوان دانش‌جو به مدل بزرگتر یا معلم نزدیک شود.

برخی از مزایای تقطیر دانش عبارتند از:

1. **کاهش پیچیدگی مدل :**مدل‌های بزرگ معمولاً دارای تعداد زیادی پارامتر هستند که نیاز به منابع محاسباتی زیادی دارند. با تقطیر دانش، می‌توان یک مدل سبکتر با دقت مشابه به دست آورد که در برخی موارد مفید است.
2. **اجتناب از بیش‌برازش :(Overfitting)** مدل‌های بزرگ ممکن است در داده‌های کمی که برای آموزش در دسترس است، به خوبی عمل کنند اما در مقابل داده‌های جدید بسیار بی‌تجربه باشند. مدل کوچکتر که با تقطیر دانش آموزش دیده است، ممکن است بهتر در مقابل داده‌های جدید عمل کند.
3. **سرعت پیش‌بینی بالا :**مدل‌های کوچکتر عموماً سریعتر در فرآیند پیش‌بینی هستند. این امر می‌تواند بسیار مهم باشد در برنامه‌ها و سیستم‌هایی که نیاز به پیش‌بینی در زمان واقعی دارند.

با کمک تقطیر دانش، مدل دانش‌جو تلاش می‌کند تا توزیع احتمالات خروجی‌های خود را به‌طور مشابه با توزیع احتمالات خروجی مدل معلم کند و در عین حال به صورت خودکار تطابق با برچسب‌ها ویژگی‌های مهم از مدل معلم یاد بگیرد.

۳.ب روند یادگیری تقطیر دانش:

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 همان مجموعه‌ی آموزشی یا یک "مجموعه انتقال "را برای آموزش مدل سنگین به کار ببریم. وقتی مدل سنگین مجموعه‌ای از 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.
بیشتر اطلاعاتی که در مورد تابع آموخته شده وجود دارد در نسبت‌هایی از احتمالات بسیار کوچک در اهداف نرم باقی می‌ماند. این اطلاعات ارزشمند هستند و ساختار شباهت موجود در داده‌ها را نشان می‌دهند (که برای مثال می‌گوید کدام ۲ شبیه ۳ و کدام شبیه ۷ به نظر می‌رسد یا کدام نژادهای سگ به هم شباهت دارند)، اما تأثیر بسیار کمرنگی روی تابع هزینه‌ی آنتروپی متقاطعطی مرحله‌ی انتقال دارند، زیرا مقادیر احتمالات به صفر خیلی نزدیک هستند.
روش تقطیر دانش یک روش آموزشی است که از یک شبکه یادگیری عمیق پیچیده به عنوان معلم (teacher) به یک شبکه کوچک‌تر به عنوان دانش‌جو (student) دانش منتقل می‌کند. این روش می‌تواند به عنوان یک راهبرد برای کاهش پیچیدگی مدل‌ها، افزایش سرعت پیش‌بینی، و یادگیری از داده‌های محدودتر مورد استفاده قرار گیرد.

ی شبکه برای تقطیر دانش:

1. **شبکه معلم:(Teacher)**
   - مدل معلم معمولاً یک شبکه عمیق و پیچیده است که با داده‌های آموزش و برچسب‌های متناظر آموزش دیده شده است.
   - خروجی‌های مدل معلم، که ممکن است احتمالات دسته‌ها یا مزایای دیگر باشند، به عنوان خروجی‌های "نرم (soft) "یا "توزیع احتمالاتی (probability distribution) "در نظر گرفته می‌شوند.
2. **شبکه دانش‌جو:(Student)**
   - مدل دانش‌جو معمولاً سبک‌تر و کوچک‌تر از مدل معلم است. تعداد کمتری پارامتر و لایه دارد.
   - خروجی‌های مدل دانش‌جو می‌توانند به عنوان خروجی‌های "نرم" تعبیه شوند، به این معنا که به جای یک برچسب دقیق، توزیع احتمالاتی از کلاس‌ها به عنوان خروجی داریم.

روند یادگیری:

1. **آموزش مدل معلم:**
   - مدل معلم با استفاده از داده‌های آموزش و برچسب‌های متناظر خود آموزش داده می‌شود.
   - خروجی‌های نرم (احتمالاتی) مدل معلم برای داده‌های آموزشی ثبت می‌شوند.
2. **آموزش مدل دانش‌جو:**
   - مدل دانش‌جو با استفاده از داده‌های آموزشی و برچسب‌های متناظر مدل معلم، آموزش داده می‌شود.
   - خروجی‌های نرم مدل دانش‌جو با خروجی‌های نرم مدل معلم مقایسه می‌شوند.
   - تابع خطا معمولاً از تفاوت توزیع احتمالاتی بین مدل معلم و دانش‌جو به عنوان خطای تقطیر (distillation loss) استفاده می‌شود.

- هدف اصلی این است که مدل دانش‌جو یاد بگیرد تا توزیع احتمالات خروجی‌های خود را به شیوه‌ای مشابه با توزیع احتمالات مدل معلم تنظیم کند.

3. **تنظیم پارامترها:**

- پارامترهای مدل دانش‌جو توسط بهینه‌سازی گرادیان کاهش می‌یابند به نحوی که توزیع احتمالات خروجی‌ها به توزیع معلم نزدیک شود.

این روش به ویژه در مواردی که داده‌های آموزش محدود هستند و یا نیاز به استفاده از مدل در محیط‌های با محدودیت محاسباتی وجود دارد، مفید است.

۳.پ تابع ضرر در student:

به منظور تقطیر دانش آموخته شده، از تابع logit (ورودی‌های لایه‌ی نهایی بیشینه‌هموار) استفاده می‌کنیم. با به حداقل رساندن مجذور تفاوت‌ها بین logit هایی که مدل سنگین و logit هایی که مدل کوچک تولید کرده‌اند، می‌توانیم از logit ها برای یادگیری مدل کوچک استفاده کنیم.

$$P_t(a) = \frac{\exp(q_t(a)/\tau)}{\sum_{i=1}^{n} \exp(q_t(i)/\tau)},$$

برای دماهای بالا (***T -> inf***) همه‌ی اقدامات، احتمال تقریباً یکسانی دارند و در دماهای پایین‌تر(***T -> 0***) ، پاداش‌هایی که بیشتر موردانتظار هستند بر احتمال تأثیر می‌گذارند. در مقادیر پایینِ پارامتر دما، احتمال اقدام با بالاترین پاداش موردانتظار نزدیک ۱ است.
در تقطیر، دمای لایه‌ی نهایی بیشینه‌هموار را افزایش می‌دهیم تا جایی که مدل سنگین مجموعه‌ی مناسبی از اهداف نرم را تولید کند. سپس همین دمای بالا را زمان آموزش مدل کوچک به کار می‌بریم تا با این اهداف نرم سازگار شود.

تابع هدف

اولین تابع هدف، آنتروپی متقاطع با اهداف نرم است. این آنتروپی متقاطع با استفاده از دمای بالای لایه‌ی بیشینه‌هموارِ مدل تقطیرشده یا دانش‌آموز (همانطور که برای تولید اهداف نرم در مدل سنگین نیز مورد استفاده قرار گرفت) محاسبه می‌شود.
دومین تابع هدف از آنتروپی متقاطع با برچسب‌های صحیح به وجود می‌آید و با استفاده از دقیقاً همان توابع logit در لایه‌ی بیشینه‌هموار مدل تقطیرشده (اما در دمای ۱) محاسبه می‌گردد.

در روش تقطیر دانش، تابع خطا یا تابع ضرر برای آموزش شبکه‌ی دانش‌جو (student) معمولاً شامل دو بخش است:

1. **تابع خطا مرتبط با برچسب‌ها:(Supervised Loss)**

- این بخش از تابع خطا با استفاده از برچسب‌های واقعی (ground truth) در داده‌های آموزش محاسبه می‌شود.
- این بخش از تابع خطا به شبکه‌ی دانش‌جو کمک می‌کند تا اطلاعات دقیقی از داده‌های آموزشی را یاد بگیرد.

2. **تابع خطا مرتبط با تقطیر:(Distillation Loss)**

- این بخش از تابع خطا براساس تفاوت توزیع احتمالات خروجی‌های شبکه‌ی دانش‌جو و معلم (teacher) محاسبه می‌شود.
- معمولاً از تفاوت میان توزیع احتمالات خروجی‌ها با استفاده از معیارهایی مانند توابع توزیع ضربان (softmax) استفاده می‌شود.

تابع خطا نهایی که برای به‌روزرسانی وزن‌های شبکه‌ی دانش‌جو استفاده می‌شود، معمولاً جمع این دو تابع خطا است. یعنی:

$$\text{Total Loss} = \text{Supervised Loss} + \lambda \times \text{Distillation Loss}$$

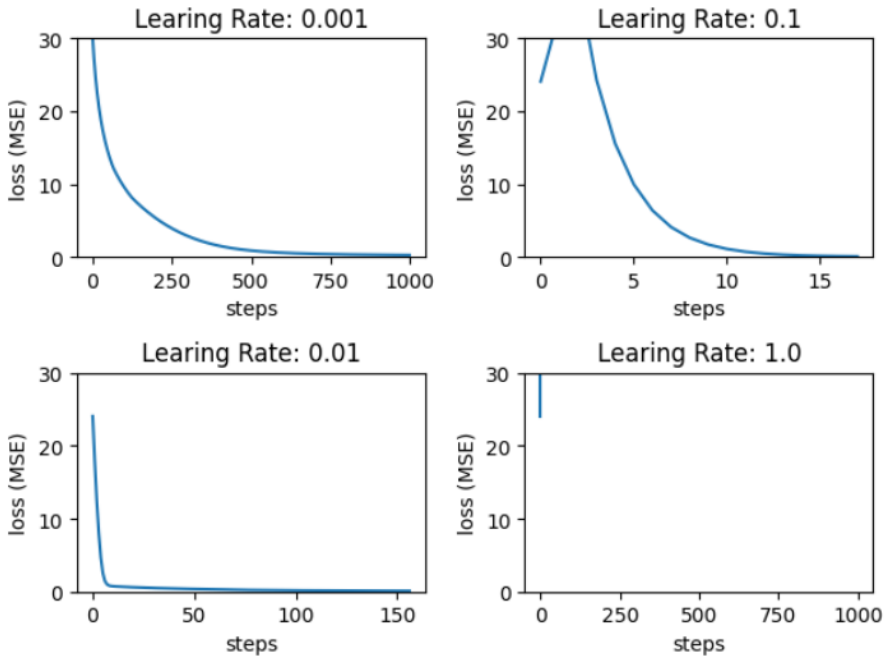
در اینجا، λ یک پارامتر است که نشان‌دهنده وزن مورد استفاده برای تابع خطا مرتبط با تقطیر است. مقدار این پارامتر معمولاً توسط کاربر یا توسعه‌دهنده تعیین می‌شود و نقش مهمی در تعادل بین دو بخش تابع خطا ایفا می‌کند. این وزن ممکن است براساس تجربیات آزمون و خطا یا تاثیر مطلوب مدل دانش‌جو تعیین شود.

منبع:
https://hooshio.com/%D8%A2%D8%B4%D9%86%D8%A7%DB%8C%DB%8C-%D8%A8%D8%A7-%D8%B1%D9%88%D8%B4-%D8%AA%D9%82%D8%B7%DB%8C%D8%B1-%D8%AF%D8%A7%D9%86%D8%B4-%D8%AC%D9%87%D8%AA-%D8%A8%D9%87%D8%A8%D9%88%D8%AF-%D8%B9%D9%85%D9%84%DA%A9/

سوال چهارم: تحلیل بهینه سازهای مختلف:

نتایج بهینه سازی SGD:

در نمودارهای زیر مقدار loss های مدل در هر مرحله آموزش به تصویر کشیده شده است. هر یک از نمودارها مربوط به یک نرخ یادگیری هستند.



تابع Loss استفاده شده در این مدل میانگین مربعات خطاها است. به همین دلیل در حالت lr=1 چون گام ها بسیار بزرگ هستند وقتی از نقطه بهینه عبور کرده، به شکل خیلی صعودی lossافزایش یافته و چون نمودار بیشتر از مقدار ۳۰ را نشان نمیدهد در گام های بعدی نمایش پیدا نکرده است.

در حالت lr=0.01, 0.001 مقدار loss کاهش پیدا کرده و به خوبی به نقطه بهینه رسیدیم اما شیب نمودار ۰،۰۱ بیشتر ایت یعنی این مقدار مناسب ترین حالت میباشد.

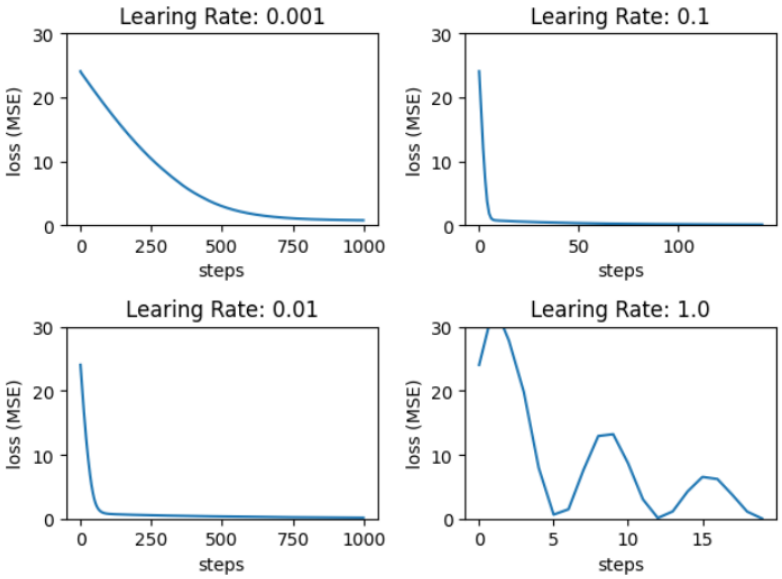
در حالت lr=0.1 ابتدا یک نقطه بهینه محلی گیرکردیم و بعد از آن خارج شده و به نقطه بهینه مطلق رسیدیم برای همین در ابتدا افزایش loss داشتیم و بعد دوباره کاهش loss داشتیم.

نتایج بهینه سازی momentum:

تغییرات زیر را در تابع انجام دادیم:

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

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

نمودار ها در تصویر زیر آمده است. ضریب مومنتوم را ۰،۱ فرض کردیم.



همانطور که قابل مشاهده است، در اینجا به ازای هر ۴ مقدار نرخ یادگیری به حالت بهینه دست پیدا کردیم و از این جهت خیلی بهتر است. تقریبا سرعت حالت های lr=0.1و lr=0.01 یکسان بوده است.

در حالت lr=0.001 دیرتر به نقطه بهینه دست پیداکردیم چون گام ها خیلی کوچتر بوده است.

در حالت lr=1 مدام به نقطه بهینه میرسیم و دوباره از آن دور میشویم چون گام ها خیلی بزرگ است اما به دلیل وجود مومنتوم میزان دورشدن از نقطه بهینه مدام کم میشود و اگر تعداد ایپاک ها بیشتر بود میتوتن پیشبینی کرد از یکجایی به بعد در نقطه بهینه میماند.

سوال پنجم:

برای حل این سوال یک مدل با لایه ورودی ۷۸۴ و لایه میانی اول ۲۵۶و لایه میانی دوم ۱۲۸ و لایه خروجی ۱۰ می‌سازیم. تابع فعال سازی دو لایه میانی relu و لایه خروجی log softmax است. تابع هزینه را crossentropyو بهینه ساز را sgd در نظر گرفتیم.

```
## Define the model
################ Your code ############

hidden_size1 = 256
hidden_size2 = 128

model = nn.Sequential(
    nn.Linear(input_size, hidden_size1),
    nn.ReLU(),
    nn.Linear(hidden_size1, hidden_size2),
    nn.ReLU(),
    nn.Linear(hidden_size2, out_size),
    nn.LogSoftmax(dim=1)
)
#######################################
```

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

```
print(model)
```

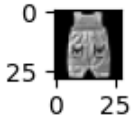
```
Sequential(
  (0): Linear(in_features=784, out_features=256, bias=True)
  (1): ReLU()
  (2): Linear(in_features=256, out_features=128, bias=True)
  (3): ReLU()
  (4): Linear(in_features=128, out_features=10, bias=True)
  (5): LogSoftmax(dim=1)
)
```

قسمت های ناقص کد تکمیل شد و چند خط کد برای نشان دادن اینکه دقت مدل و مقدار تابع ضرر روی دیتای آموزش در هر ایپاک چه قدر است به آن اضافه کردیم:

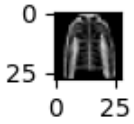
```
Epoch 1/10, Training Loss: 0.8643, Training Accuracy: 72.00%
Epoch 2/10, Training Loss: 0.5090, Training Accuracy: 82.16%
Epoch 3/10, Training Loss: 0.4539, Training Accuracy: 83.99%
Epoch 4/10, Training Loss: 0.4203, Training Accuracy: 85.18%
Epoch 5/10, Training Loss: 0.3953, Training Accuracy: 86.08%
Epoch 6/10, Training Loss: 0.3776, Training Accuracy: 86.57%
Epoch 7/10, Training Loss: 0.3620, Training Accuracy: 87.09%
Epoch 8/10, Training Loss: 0.3506, Training Accuracy: 87.48%
Epoch 9/10, Training Loss: 0.3395, Training Accuracy: 87.85%
Epoch 10/10, Training Loss: 0.3289, Training Accuracy: 88.11%
```

دقت مدل روی دیتای تست حدود ۸۶ درصد شد و برخی از نمونه ها اشتباه تشخیص میداد.

Actual: Trouser, Predicted: Trouser

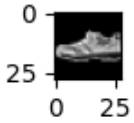


Actual: Coat, Predicted: Coat

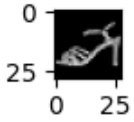


Actual: T-Shirt, Predicted: T-Shirt

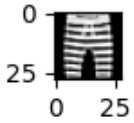
Actual: Sneaker, Predicted: Sneaker



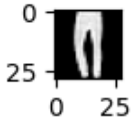
Actual: Sandal, Predicted: Sandal



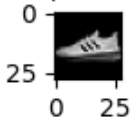
Actual: Trouser, Predicted: Trouser



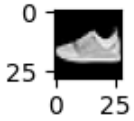
Actual: Trouser, Predicted: Trouser



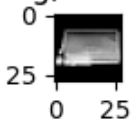
Actual: Sneaker, Predicted: Sneaker



Actual: Sneaker, Predicted: Sneaker



Actual: Bag, Predicted: Bag



سوال ششم:

ب)ایجاد بیش برازش در مدل قبلی

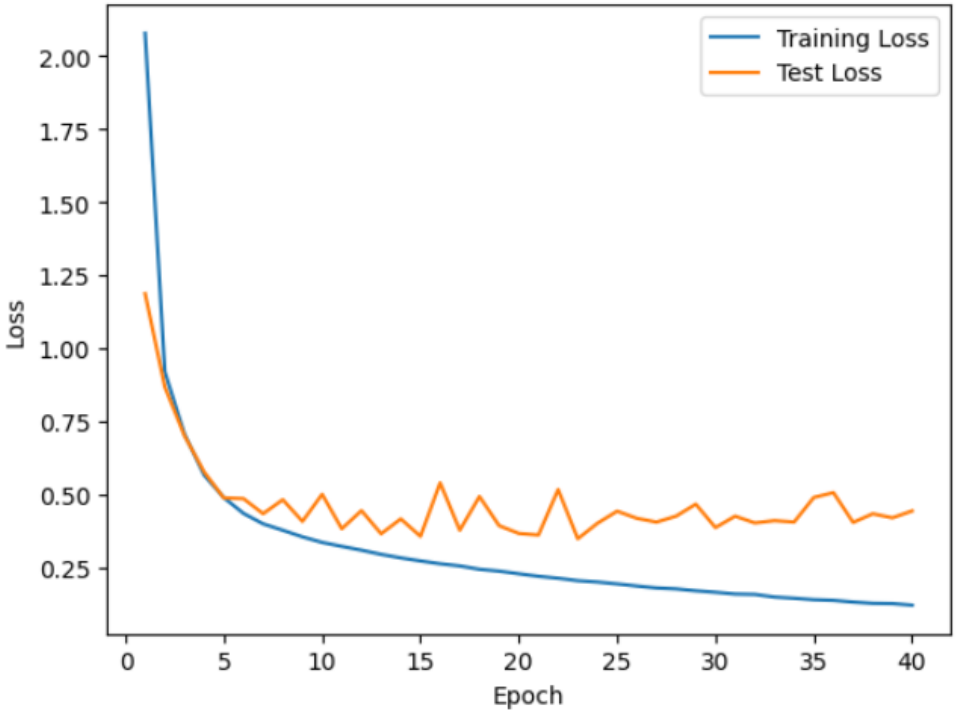
مدل قبلی بیش برازش نداشت چون دقت دیتای آموزشی و تست تقریبا مشابه بود. یکی از راه های ایجاد بیش برازش این است که مدل را عمیق تر کنیم تا پیچیده شود البته نه آنقدر عمیق که آموزش آن هم دچار مشکل شود. من تصمیم گرفتم از ۴ لایه استفاده کنم. همچنین تعداد ایپاک را افزایش دادیم تا مدل داده های آموزشی را تقریبا حفظ کند.

```
Sequential(
  (0): Linear(in_features=784, out_features=512, bias=True)
  (1): ReLU()
  (2): Linear(in_features=512, out_features=128, bias=True)
  (3): ReLU()
  (4): Linear(in_features=128, out_features=64, bias=True)
  (5): ReLU()
  (6): Linear(in_features=64, out_features=32, bias=True)
  (7): ReLU()
  (8): Linear(in_features=32, out_features=10, bias=True)
  (9): LogSoftmax(dim=1)
)
```

در بخش آموزش کد قطعه کدی نیاز داشتیم تا نمودار Loss و دقت را برحسب ایپاک رسم کند برای همین تغییراتی در بخش آموزش دادیم.

نتیجه نهایی به صورت زیر شد:

```
Epoch 40/40, Training Loss: 0.1215, Training Accuracy: 95.52%
Test Loss: 0.4439, Test Accuracy: 88.07%
```



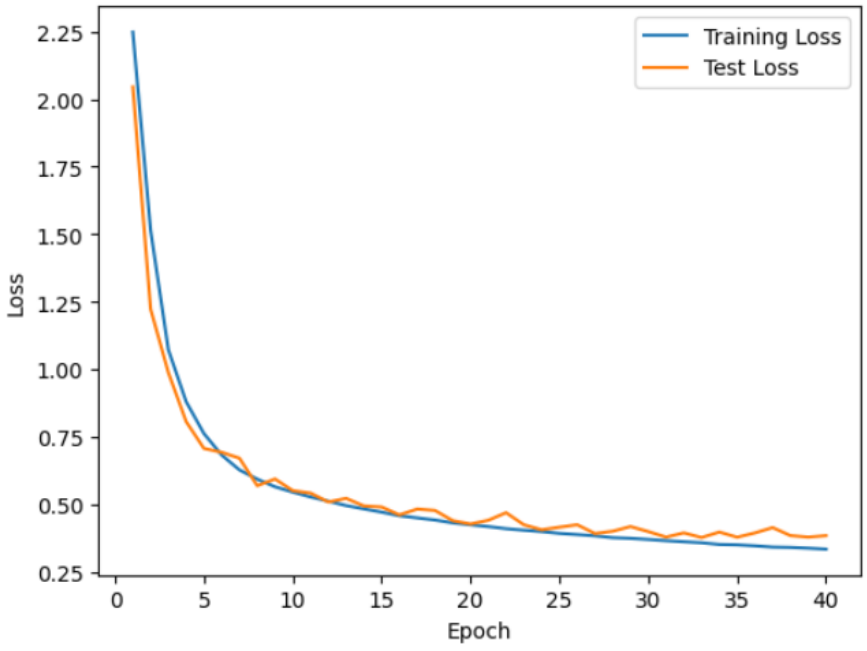
پ) رفع بیش برازش با داده افزایی

یک ترنسفورمر برای اضافه کردن دیتا ها با تغییرات مورد نظر نوشتیم سپس دیتا ست آموزش و تست و دیتا لودر را آپدیت کردیم. بخش تعریف مدل و آموزش و تست به همان شکل باقی ماند فقط در آموزش و تست از دیتالودرهای جدید استفاده شد.

```
# Define data transformations for data augmentation
transform = transforms.Compose([
    transforms.RandomHorizontalFlip(),
    transforms.RandomRotation(10),
    transforms.ToTensor(),
])

# Update your data loaders with the new transformations
trainset_c = torchvision.datasets.FashionMNIST(root='./data', train=True, download=True, transform=transform)
testset_c = torchvision.datasets.FashionMNIST(root='./data', train=False, download=True, transform=transforms.ToTensor())

trainloader_c = torch.utils.data.DataLoader(trainset_c, batch_size=64, shuffle=True)
testloader_c = torch.utils.data.DataLoader(testset_c, batch_size=64, shuffle=False)

DAmodel = nn.Sequential(
    nn.Linear(input_size, hidden_size1),
    nn.ReLU(),
    nn.Linear(hidden_size1, hidden_size2),
    nn.ReLU(),
    nn.Linear(hidden_size2, hidden_size3),
    nn.ReLU(),
    nn.Linear(hidden_size3, hidden_size4),
    nn.ReLU(),
    nn.Linear(hidden_size4, out_size),
    nn.LogSoftmax(dim=1)
)

criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(DAmodel.parameters(), lr=0.01)
```

تاحد قابل توجهی بیش برازش روی دیتای آموزشی کمتر شد و دقت در آموزش و تست نسبت به قبل به هم نزدیک تر شدند.

```
Epoch 40/40, Training Loss: 0.3340, Training Accuracy: 87.79%
Test Loss: 0.3838, Test Accuracy: 85.87%
```



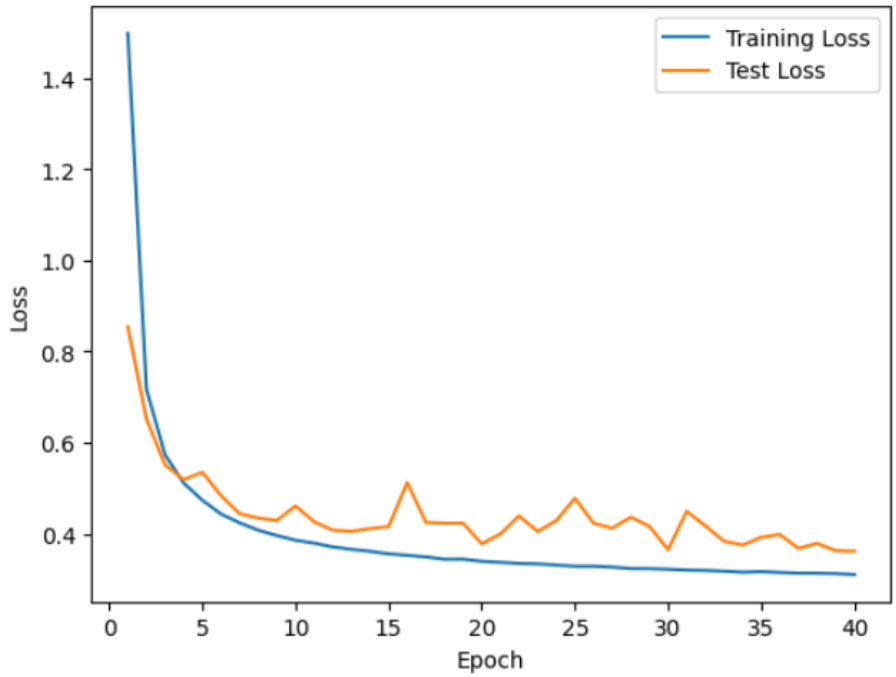
ت) رفع بیش برازش با استفاده از منظم سازی L2

از مدل اولیه و دیتاست بخش ب استفاده کردیم و فقط در آپتیمایز ضریب مربوط به L2 را هم دادیم.

```
# Define the optimizer with L2 regularization (weight decay)
optimizer = optim.SGD(modelL2.parameters(), lr=0.01, weight_decay=0.01)
```

نتیجه به صورت زیر شد:

```
Epoch 40/40, Training Loss: 0.3110, Training Accuracy: 89.29%
Test Loss: 0.3621, Test Accuracy: 87.18%
```



همانطور که از نمودار میتوان فهمید نمودار خطای تست در حالت داده افزایی، هموار تر یا اصطلاحا اسموس تر بود و به نمودار خطای داده ی ترین نزدیک تر بود، با ابن حال این روش منظم سازی L2 هم تا حد خوبی جلوی بیش برازش را گرفته است.

ث) رفع بیش برازش با ترکیبی از داده افزایی،منظم سازی و دراپ اوت

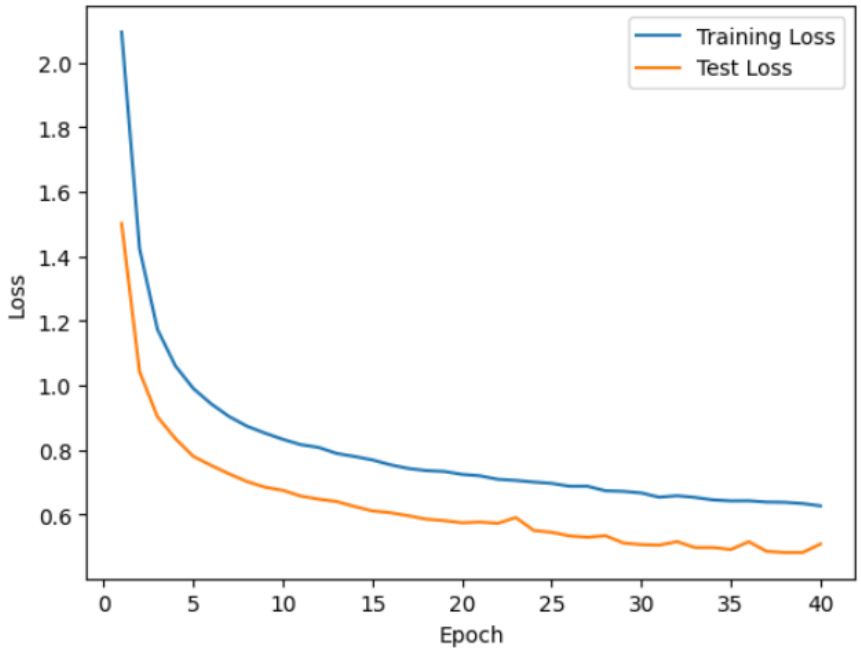
در مدل بعد از هر لایه relu یک لایه دراپاوت با احتمال ۰٫۵ اضافه میکنیم.

```
model_combined = nn.Sequential(
    nn.Linear(input_size, hidden_size1),
    nn.ReLU(),
    nn.Dropout(dropout_prob),
    nn.Linear(hidden_size1, hidden_size2),
    nn.ReLU(),
    nn.Dropout(dropout_prob),
    nn.Linear(hidden_size2, hidden_size3),
    nn.ReLU(),
    nn.Dropout(dropout_prob),
    nn.Linear(hidden_size3, hidden_size4),
    nn.ReLU(),
    nn.Dropout(dropout_prob),
    nn.Linear(hidden_size4, out_size),
    nn.LogSoftmax(dim=1)
)
```

ترنسفورمر برای داده افزایی و ضریب L2 مانند بخش های قبل تمرین است و تغییر نکرده فقط هر ۳ حالت اعمال شده است. این ضریب ها بهترین حالت بود مثلا برای L2 ۰,۰۰۱ هم امتحان شد اما خوب نبود و از 0.01 استفاده میکنیم.

نتایج به صورت زیر درآمد:

```
Epoch 40/40, Training Loss: 0.6265, Training Accuracy: 80.01%
Test Loss: 0.5084, Test Accuracy: 82.48%
```



همانطور که مشخص شده است مشکل اورفیت کاملا رفع شده است و با توجه به کم بودن دقت در دیتاست آموزش نسبت به تست، مشخص میشود برای گرفتن نتیجه بهتر و داشتن مدلی که دقتش روی تست ها به ۹۰ برسد باید مدل پیچیده تری انتخاب کنیم.