سوال ۱-

الف) بیش برازش: این مشکل فقط مختص به شبکه های عصبی نیست و میتواند در هر روش یادگیری ماشین رخ دهد. معنای آن این است که مدل ما (برای مثال شبکه عصبی) آن قدر خوب دیتای آموزش را یاد گرفته که شروع کرده به حفظ کردن نمونه‌های خاص به جای اینکه الگوی پنهان درون آنها را کشف کند و در نتیجه در generalization ضعف پیدا کرده است. در واقع اگر مدل ما ظرفیت بالایی داشته باشد و ما جلوی آموزش را نگیریم، مدل ما به overfit شدن نزدیک و نزدیک‌تر می‌شود. نشانه‌ی بارز این موضوع زمانی هست که خطای دادهٔ آموزش بسیار کم است ولی خطای دادهٔ تست و اعتبارسنجی خیلی بالاست و با خطای آموزش بسیار تفاوت دارد، که این نشان‌دهنده عدم تعمیم‌دهی است. برای حل این مشکل میتوان آموزش شبکه را زودتر متوقف کرد یا از روشهایی مثل weight decay استفاده کرد.

کم‌برازش: این مشکل نیز صرفا به شبکه عصبی اختصاص ندارد. اما در این حالت، مدل ما قدرت کافی برای کشف کردن الگوهای داده‌های آموزش را ندارد که میتواند بخاطر ساده بودن مدل یا شبکه عصبی باشد. اگر خطای آموزش و تست و اعتبارسنجی، همگی بزرگ باشند، به معنای underfit بودن مدل است. این اتفاق میتواند بخاطر کم بودن دیتای آموزش هم باشد، البته در این حالت مدل ما میتواند همان دیتای کم را حفظ کند که میشود همان overfit شدن. راه مقابله با این مشکل، پیچیده‌تر کردن مدل یا در مبحث شبکه‌های عصبی، بیشتر کردن تعداد لایه‌ها و نورونها است.

ب) دو حالت را در نظر می‌گیرم:

حالت اول: وجود دیتای اعتبارسنجی

در این حالت میتوان دقت و خطای مدل را بر روی دادهٔ آموزش و دادهٔ اعتبارسنجی محاسبه کرد. اگر این دو فاصله زیادی با یکدیگر داشتند، این میتواند یک نشانهٔ بیش‌برازش باشد. همچنین میتوان نمودار خطا را کشید و این موضوع را روی آن بررسی کرد، به طوریکه خطای اعتبارسنجی پس از کاهش تا یک مقداری، شروع به افزایش میکند.

حالت دوم: عدم وجود دیتای اعتبارسنجی

در این حالت میتوان cross-validation انجام داد، به طوری که دیتای آموزش را به چندین fold تقسیم کنیم، روی یکی از فولدها اعتبارسنجی و روی بقیه آموزش را انجام دهیم و خطای آموزش و اعتبارسنجی را گزارش دهیم. دفعه بعد یک فولد دیگر را به اعتبارسنجی اختصاص داده و بقیه را به آموزش و باز هم خطاها را

گزارش می‌دهیم. اگر در این گزارشات در اکثر موارد، تفاوت بین دو خطا زیاد بود، میتواند نشانهٔ بیش‌برازش باشد.

پ)

خروجی بخش آموزش:

1.6	0	0	1.9
0	2.5	2.5	0
0	3.2	3.7	0
1.3	0	0	1.2

خروجی بخش آزمون:

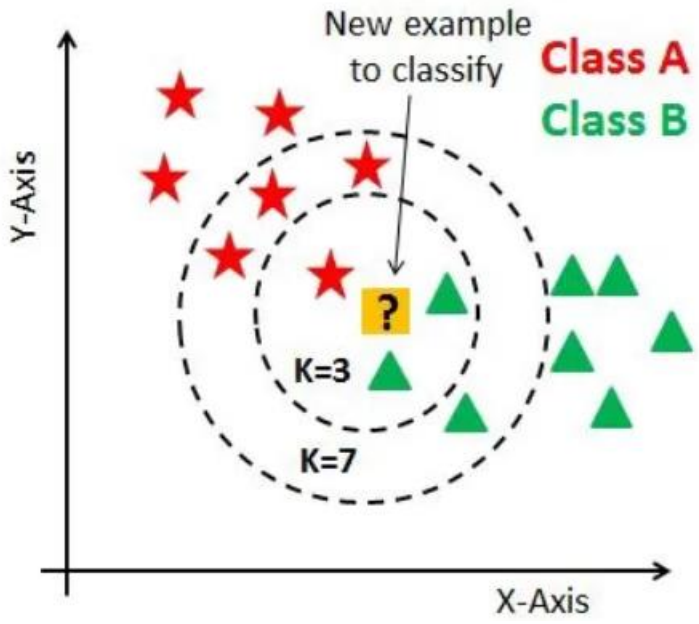
مقدار p طبق ماسک برابر است با: 0.5

0.8	-0.35	-0.1	0.95
-1.15	1.25	1.25	-0.45
-0.25	1.6	1.85	-0.2
0.65	-0.2	-1.3	0.6

سوال ۲-

الف)

بررسی واریانس: با افزایش مقدار K، مدل ما محدودهٔ بزرگتری را برای تصمیم خودش در نظر میگیرد و این باعث می‌شود حساسیت آن به برخی از داده‌های آموزش که نویزی هستند، کاهش پیدا کند. برای مثال در عکس زیر وقتی این مقدار برابر با ۳ بود، کلاس ب انتخاب شد ولی با افزایش این مقدار، کلاس آ انتخاب شد. در واقع ممکن است، در فاصله کم از دادهٔ تست، چندین دیتا از کلاس مخالف حضور داشته باشند که دادهٔ پرت حساب میشوند، در حالیکه که مقدار K کوچک به آنها اهمیت زیادی می‌دهد.



بایاس: مقدار K بزرگ به معنای محدودهٔ تصمیم (decision boundary) بزرگ است. این بدین معناست که مدل ما یک مدل ساده است، زیرا مسئله را بسیار ساده فرض کرده است. در حالت extreme فرض کنید که مقدار K آنقدر بزرگ است که کل دیتای آموزش را در نظر می‌گیرد، پس میگوید بین همه این دیتای آموزش هر کدام که تعداد بیشتری بود را انتخاب کن، در این حالت، بدون توجه به دادهٔ تست جدید، همه را به یک کلاس اختصاص میدهد و آن هم کلاسی است که در دیتای آموزش بیشتر حضور داشته است. در نتیجه Bias مدل افزایش پیدا میکند زیرا قادر به انتخاب درست دسته نخواهد بود.

در نتیجه برای انتخاب مقدار مناسب K با یک trade-off بین بایاس و واریانس روبرو هستیم.

ب)

مورد اول: صحیح است. اگر در منظم سازی، مقدار پارامتر منظم سازی را زیادی بزرگ در نظر بگیریم، جلوی یادگیری مدل را می‌گیرد، زیرا اجازه نمیدهد مدل با دست باز، پارامترها را به گونه‌ای که الگوها را حفظ کنند، تنظیم کند.

مورد دوم: غلط است. اضافه کردن ویژگی بیشتر به معنای پیچیده‌تر کردن مدل است و این باعث بالا رفتن احتمال بیش‌برازش می‌شود.

مورد سوم: غلط است. با زیاد کردن ضریب منظم‌سازی، احتمال بیش‌برازش کمتر و احتمال کم‌برازش بیشتر می‌شود. هرچه بیشتر به مقدار وزنها در تابع ضرر اهمیت بدهیم، مدل کمتر میتواند از ظرفیت خود برای حفظ جزئیات استفاده کند و در نتیجه احتمال overfit شدن کمتر می‌شود.

پ)

- $W_{exp1} = [0.26, 0.25, 0.25, 0.25]$
- $Wexp2 = [1, 0, 0, 0]$
- $W_{exp3} = [13.3, 23.5, 53.2, 5.1]$
- $W_{exp4} = [0.5, 1.2, 8.5, 0]$

در آزمایش اول، وزنها در یک محدوده و نزدیک هم هستند و این ویژگی L2 regularization است که جلوی بزرگ شدن بیش از حد همهٔ وزنها را میگیرد.

با فرض اینکه مقادیر اولیه وزنها صفر نبوده‌اند، مورد دوم و چهارم که مقدار صفر را شامل می‌شوند، در صورتی که از منظم سازی استفاده کرده باشند، از L1 استفاده کرده‌اند زیرا در L1 به دلیل فرمولی که دارد، میتواند مقدار یک وزن را صفر بکند. البته برای مورد چهارم، وزنها نرمال نیستند و وزن سوم فاصله زیادی با بقیه دارد و احتمال در این مورد از منظم سازی استفاده نشده است، البته این خیلی بستگی به مقدار ضریب منظم سازی دارد و نمیتوان قطعی گفت. برای آزمایش دوم میتوان گفت که از L1 استفاده شده است، زیرا یکی از وزنها با بقیه خیلی فاصله دارد و sparse بودن یکی از ویژگیهایی است که L1 به وزنها میدهد و خیلی از آنها را صفر میکند.

برای مورد سوم میتوان گفت که از منظم سازی استفاده نشده است زیرا فاصله وزنها بسیار زیاد است و این مطابق با L1 و L2 Regularization نمیباشد.

آزمایش چهارم هم احتمالا L1 است زیرا یکی از وزنها با بقیه خیلی فاصله دارد که مشابه با آزمایش دوم، یکی از ویژگیهایی است که L1 Regularization باعث آن میشود.

برای این سوال، از claude استفاده کردم و از پرامپت زیر بهره بردم:

c) Suppose we have a model and to prevent overfitting, we want to regularize L1 and L2.
let's use For this purpose, four experiments were performed and the results obtained are as follows. With
Paying attention to these results, specify which regularization was used in each experiment (explain reason for your answer).
• Wexp1 = [0.26, 0.25, 0.25, 0.25]
• W exp2 = [1, 0, 0, 0]
• Wexp3 = [13.3, 23.5, 53.2, 5.1]
• Wexp4 = [0.5, 1.2, 8.5, 0]

جواب claude:

Here is my analysis of the regularization used in each experiment:

Experiment 1: L2 regularization

The weights are relatively small and similar in value. This is consistent with L2 regularization which discourages large weights and pushes all weights to be small.

Experiment 2: L1 regularization

One weight is much larger than the others, close to 1, while the rest are 0. This sparse pattern is characteristic of L1 regularization which drives many weights to exactly 0 .

Experiment 3: No regularization

The weights have very different large values, some orders of magnitude apart. This is inconsistent with L1 or L2 regularization which would produce more uniform weights. So this experiment likely did not use any regularization.

Experiment 4: L1 regularization

One weight is much larger than the others, similar to experiment 2. While not exactly 0, the smaller weights are relatively small, indicating L1 sparsity inducing behavior.

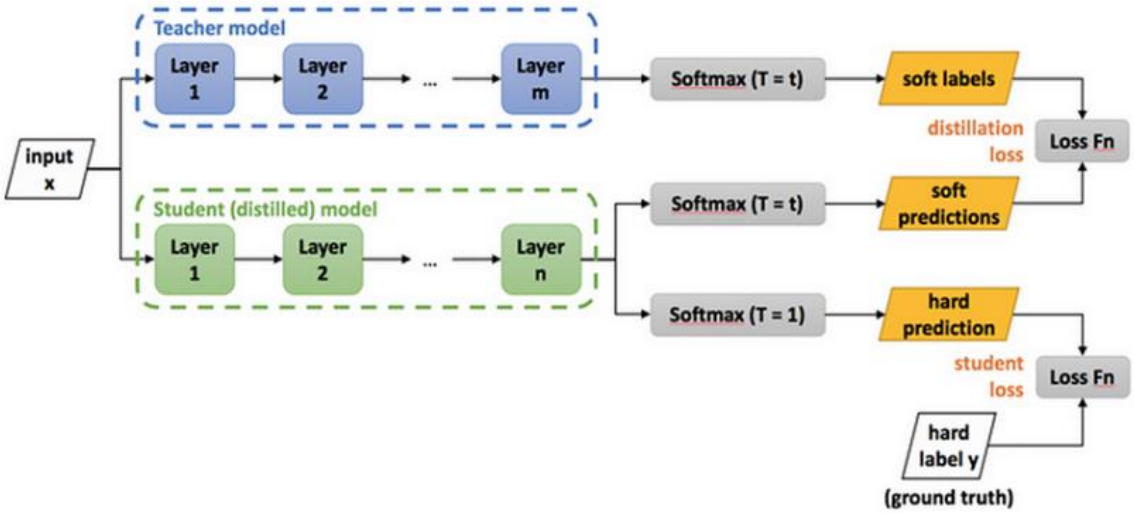
In summary, based on the characteristic weight patterns produced by each type of regularization, experiment 1 used L2, experiments 2 and 4 used L1, and experiment 3 did not use any regularization.

اما در مورد اول به احتمال زیاد از منظم سازی استفاده شده است زیرا مقادیر وزنها به یکدیگر نزدیک هستند. اما اینکه از کدام نوع منظم سازی استفاده شده است، میتواند هر کدام از L1 یا L2 باشد.

سوال ۳-

الف) فرایند تقطیر دانش، به معنای انتقال دانش از یک مدل که روی یک مسئله نتیجه خوب گرفته (معلم) به یک مدل که میخواهد روی همان مسئله یا یک مسئله شبیه آن یادگیری انجام دهد (دانش‌آموز)، می‌باشد. این کار به مدل دانش‌آموز به شکلهای مختلف کمک میکند. برای مثال اگر وزنهای یادگرفته شده را از مدل معلم و به مدل دانش‌آموز انتقال دهیم، یادگیری مدل دانش‌آموز سریعتر می‌شود و سریعتر به دقتهای بالا می‌رسد. یا در مورد hard labels میدانیم اگر در دیتای آموزش، برچسب یک دیتا را اشتباه به مدل بدهیم، صفر و یکی بودن برچسبها باعث میشود که مدل ما وزنها را خیلی جریمه کند. پس بجای آن از soft labels استفاده میکنند که یک احتمال کوچک برای بقیه کلاسها هم در نظر می‌گیرد. یکی از انواع فرایندهای تقطیر دانش، اعمال آن در سطح پاسخ یا خروجی است که از پاسخ مدل معلم برای محاسبه تابع ضرر در خروجی دانش‌آموز استفاده می‌شود.

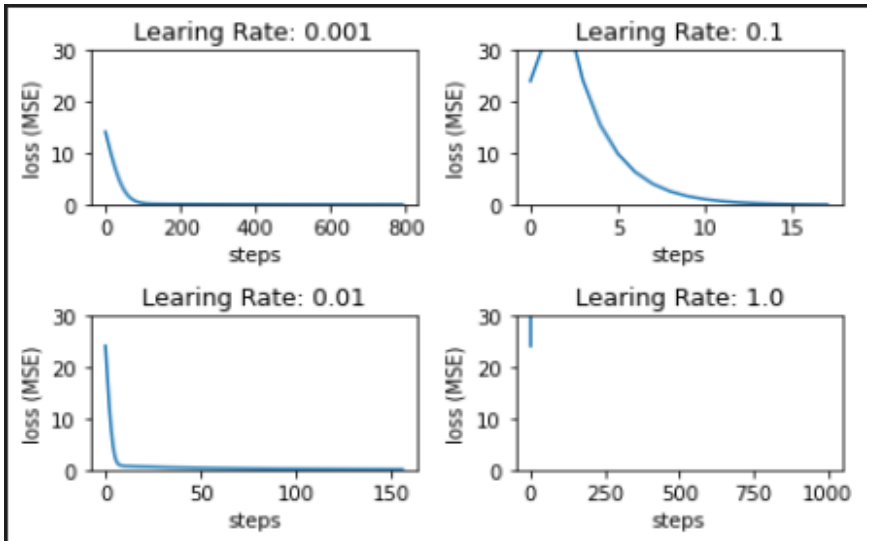
ب)



در این تصویر، مدل معلم از قبل آموزش دیده است و دقت بالایی دارد. میخواهیم مدل دانش‌آموز را آموزش دهیم. ورودی را به هردوی مدلها میدهیم. هر کدام یک خروجی میدهند. خروجی معلم و دانش‌آموز را به تابع ضرر میدهیم و این تابع از خروجی دانش‌آموز به عنوان پاسخ مدل و از خروجی معلم، به عنوان ground truth برای محاسبه خطا استفاده میکند.

پ) وزنها را بر اساس تابع ضرری که از خروجی معلم به عنوان ground truth استفاده میکند (soft labels)، آپدیت میکنیم، زیرا همانطور که گفتیم، میخواهیم از soft labels برای جریمه کردن استفاده کنیم تا برچسب اشتباه، باعث یک جریمه سنگین نشود.

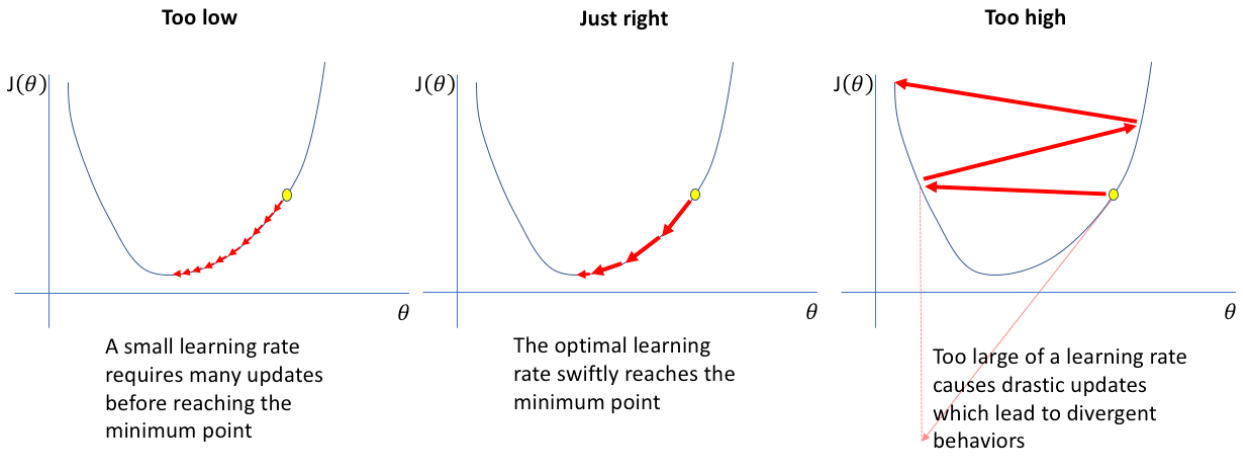
سوال ۴-



دو نوع بهینه‌ساز در این کد بررسی شده است. اولی SGD است که با استفاده از میانگین loss ها بر روی یک batch و با فرمول روبرو، پارامترها را آپدیت می‌کند. `a.data -= a.grad * lr`

تحلیل نمودارها:

ابتدا به عکس زیر توجه کنید که تاثیر نرخ یادگیری بر روی روند آموزش مدل را نشان می‌دهد.

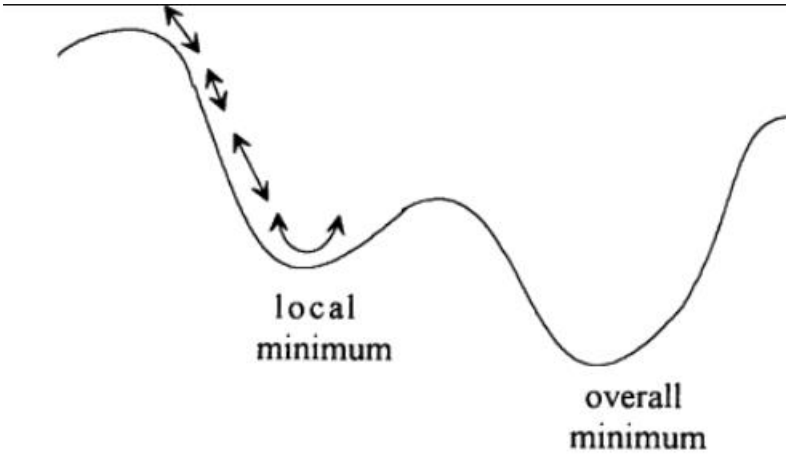


اگر این مقدار خیلی بزرگ باشد، مدل به صورت واگرا عمل کرده و از نقطه بهینه دورتر و دورتر میشود، اتفاقی که در نمودار پایین سمت راست برای lr=1 افتاده است. در حالت بعدی که lr=0.1 است، ابتدا مقدار loss زیاد میشود و سپس سیر نزولی پیدا می‌کند. این میتواند به این دلیل باشد که ابتدا مدل ما در نزدیکی یک مینیمم محلی واقع بوده و با این نرخ یادگیری، از این مینیمم محلی فرار کرده و همچنین به نقطه‌ای رسیده که مقدار تابع ضرر برای آن مقدار بیشتری است، اما حالا که از مینیمم محلی قبلی فرار کرده، میتواند در یک مینیمم دیگر(که میتواند آن هم محلی باشد) همگرا شود که این اتفاق می‌افتد. به طور کلی، اگر مقدار نرخ یادگیری، خیلی بزرگ نباشد، در حالت پایانی میتواند به یک مینیمم (محلی یا جهانی) همگرا شود. برای مقادیر بزرگتر نرخ یادگیری، این همگرایی زودتر رخ میدهد. برای lr=0.001 پس از ۲۰۰ epoch و برای lr=0.01 پس از ۵۰ epoch و برای lr=0.1 پس از ۱۵ epoch، این همگرایی صورت میگیرد. (اعداد ذکر شده به صورت تقریبی از روی نمودار خوانده شد)

برای اینکه بتوانیم از مینیمم های محلی فرار کنیم، از نسخه momentum استفاده میکنیم. شکل زیر را در نظر بگیرید. اگر از نقطه بالا چپ شروع کنیم، به درون مینیمم محلی میرویم و همانجا می‌مانیم. اما اگر یک توپ

را فرض کنید که از این نقطه رها شود، به احتمال زیاد، با سرعتی که بدست می‌آورد، از سمت دیگر این مینیمم محلی بالا میرود و احتمالا در سرازیری مربوط به مینیمم جهانی قرار میگیرد و به این نقطه همگرا میشود.

برای شبیه‌سازی چنین چیزی در شبکه عصبی و فرایند بهینه سازی، از momentum استفاده می‌کنیم.

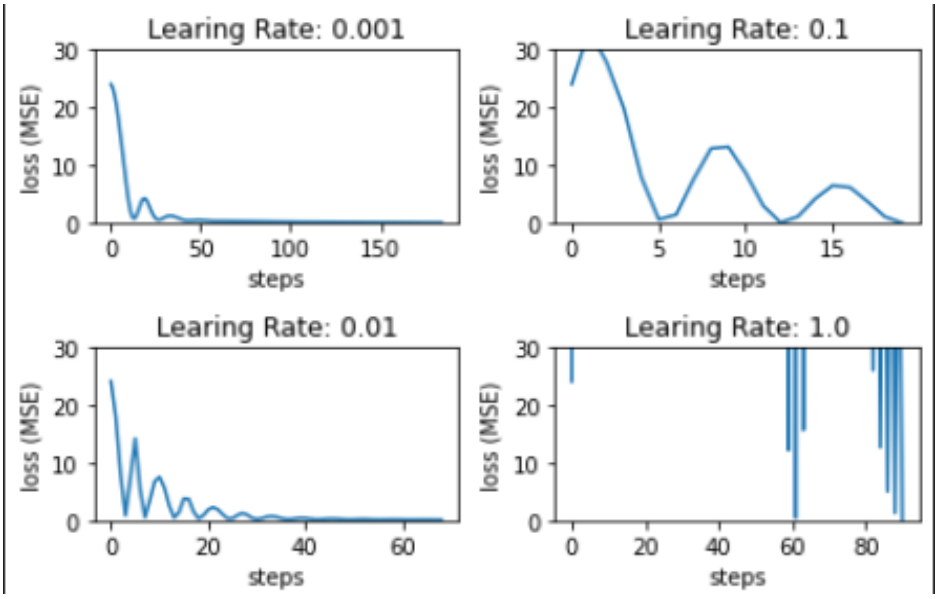


$$v_{t+1} = \rho v_t + \nabla f(x_t)$$
$$x_{t+1} = x_t - \alpha v_{t+1}$$

```
vx = 0
while True:
  dx = compute_gradient(x)
  vx = rho * vx + dx
  x -= learning_rate * vx
```

یک ترم برای سرعت نیز اضافه میکنیم، البته با یک ضریب که نقش اصطکاک را دارد.

همانطور که انتظار داریم، در این حالت، مدل به یک مینیمم میرسد و از آن مینیمم فاصله میگیرد، اگر عمق این مینیمم به اندازه کافی باشد، پس از چند بار نزدیک شدن و دور شدن، نهایتا در آن آرام می‌گیرد.



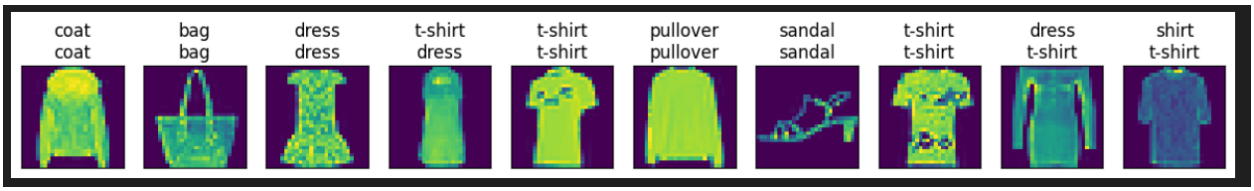
برای نرخ یادگیری برابر با ۱ ، همگرا نشده است ولی برای مقادیر دیگ، این اتفاق رخ داده. سریعترین همگرایی برای lr=0.1 است و دیرترین برای lr=0.001 می‌باشد.

سوال ۵-

الف) برای این بخش، از ساختار زیر برای مدل خودم استفاده کردم.

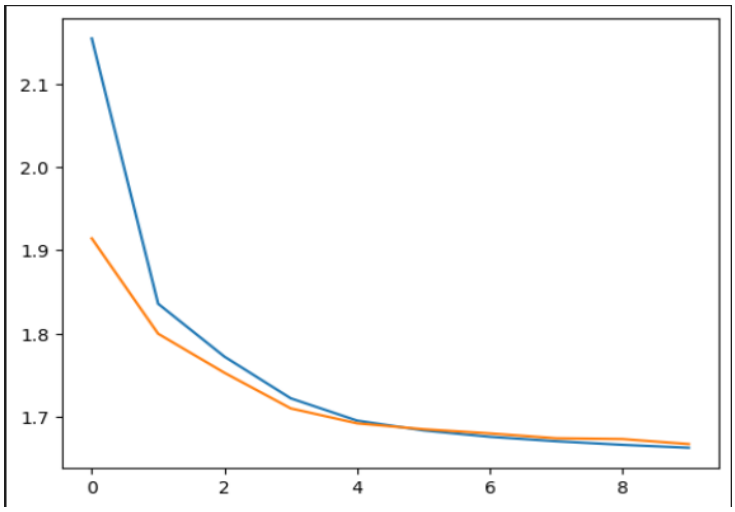
```
model1 = nn.Sequential(
  nn.Flatten(),
  nn.Linear(input_size, 128, device=device),
  nn.ReLU(),
  nn.Linear(128, 64, device=device),
  nn.ReLU(),
  nn.Linear(64, out_size, device=device),
  nn.Softmax()
)
```

سه لایه خطی که تعداد نورونها را هر بار کاهش دادم و برای لایه آخر تعداد نورون به تعداد کلاسها گرفتم تا به کمک تابع softmax احتمال هر کلاس را استخراج کند. همچنین مدل را بر روی GPU بردم ولی Colab تمام ظرفیت GPU را در اختیار نمیگذاشت و با حالت بدون GPU فرقی نداشت و آموزش خیلی طول میکشید.



مدل حاصل، برخی از لباسها که شبیه هم هستند را به اشتباه دسته بندی کرد. مثل shirt و t-shirt یا dress و t-shirt.

نمودار loss به صورت زیر شد:



و مقدار accuracy زیر:

```
accuracy_test = 0
for images,labels in testloader:
  output = model1(images)
  output = torch.argmax(output, axis=-1)
  accuracy_test += torch.sum(output == labels).item()
accuracy_train = 0
for images,labels in trainloader:
  output = model1(images)
  output = torch.argmax(output, axis=-1)
  accuracy_train += torch.sum(output == labels).item()
print(f"Test Accuracy: {accuracy_test / 10000}")
print(f"Train Accuracy: {accuracy_train / 60000}")
```

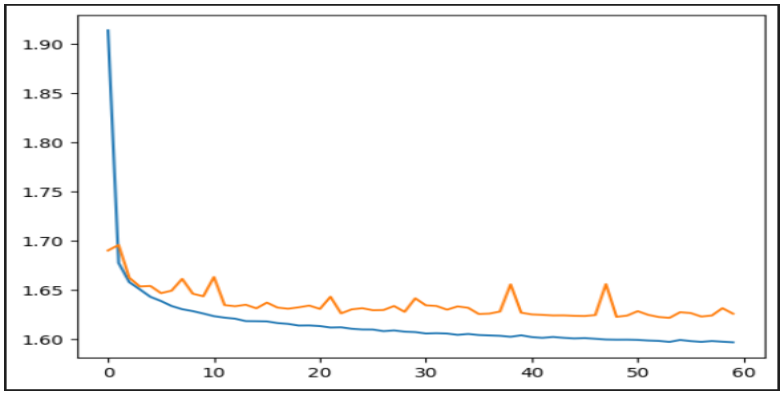
```
Accuracy: 0.8012
```

ب) میخواهیم یک مدل که overfit شده است، ایجاد کنیم. برای این کار میتوان تعداد لایه‌ها و تعداد نورونهای هر لایه را افزایش داد و تعداد epoch های آموزش را نیز زیاد کرد.

مدلی که برای این منظور گذاشتم، به شکل زیر بود:

```
model = nn.Sequential(
  nn.Flatten(),
  nn.Linear(input_size, 1024),
  nn.ReLU(),
  nn.Linear(1024, 512),
  nn.ReLU(),
  nn.Linear(512, 512),
  nn.ReLU(),
  nn.Linear(512, 512),
  nn.ReLU(),
  nn.Linear(512, out_size),
  nn.Softmax()
)
```

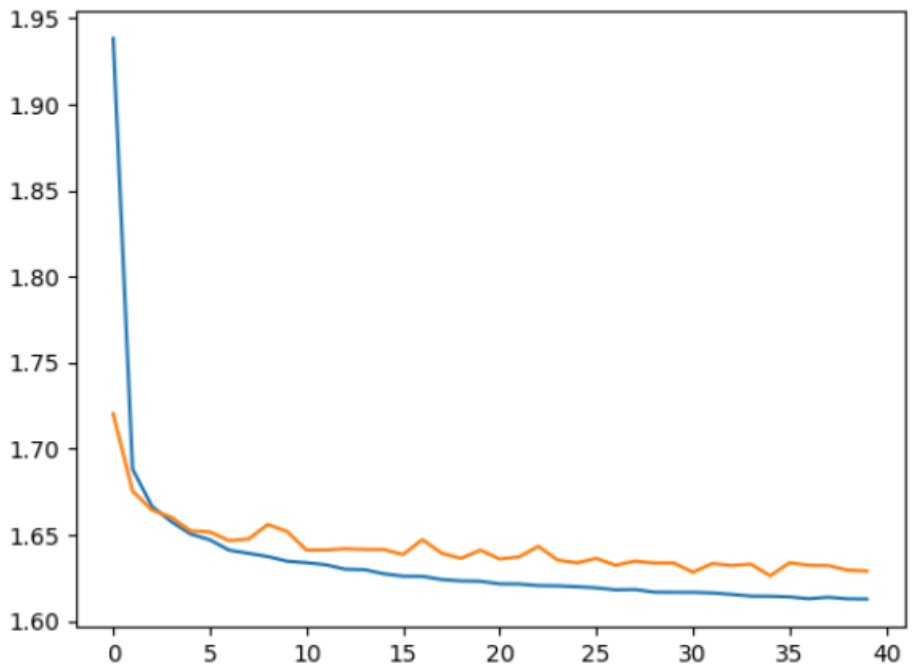
و نمودار loss به صورت زیر شد:



و مقدار accuracy زیر:

```
Test Accuracy: 0.8348
Train Accuracy: 0.85815
```

پ) این بار از داده‌افزایی استفاده کردم. با یک احتمالی روشنایی عکس را زیاد میکنم و با یک احتمال نیز، عکس را می‌چرخانم. این کار برای جلوگیری از overfitting انجام می‌شود.



مقدار آخرین loss برای حالت ب:

```
Training loss: 1.5970206164093668          Validation loss: 1.626015244775517
```

مقدار آخرین loss پس از اضافه کردن data augmentation:

```
Training loss: 1.6127475684385564          Validation loss: 1.6290583071435334
```

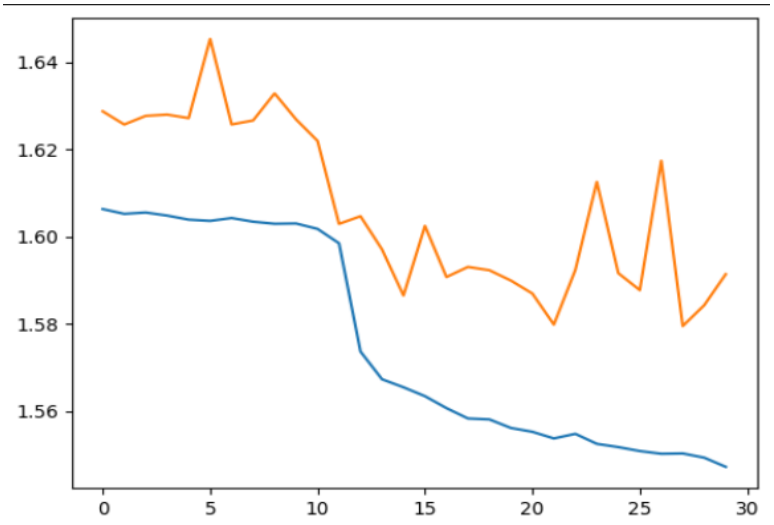
همانطور که مشخص است، مقدار loss برای دادهٔ آموزش به دادهٔ اعتبارسنجی نزدیک شده است ولی این به میزانی نبود که مقادیر accuracy تغییری بکند، همچنین این مقدار بر روی داده اعتبارسنجی تغییر زیادی نداشت.

مقدار accuracy:

```
Test Accuracy: 0.8307
Train Accuracy: 0.8522666666666666
```

ت) این بار از L2 regularization برای جلوگیری از بیش‌برازش یا overfitting استفاده کردم.

نمودار تابع ضرر به صورت زیر شد:



و مقادیر آخر loss نسبت به حالت قبل خیلی بهتر شد و دقت مدل هم بر روی train هم بر روی test بالا رفت.

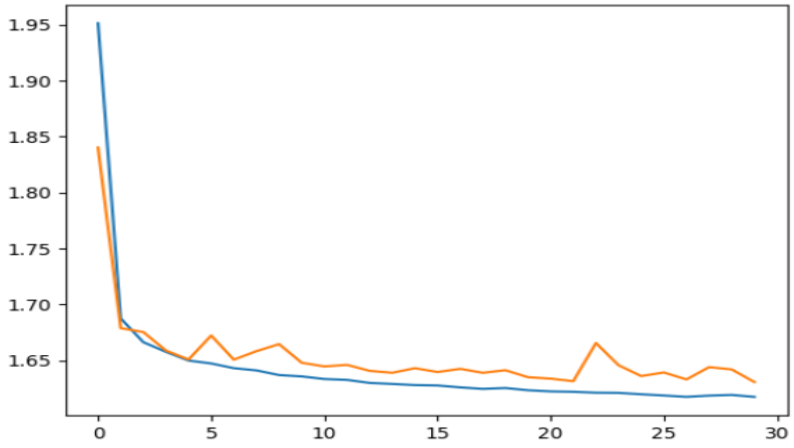
مقدار accuracy:

```
Test Accuracy: 0.8695
Train Accuracy: 0.90645
```

ث) حالا از ترکیب چندین روش جلوگیری از overfitting مثل dropout,weight decay,data augmention استفاده کردم ولی نتیجه خوب نبود، زیرا استفاده از چندین روش به طور همزمان، برای مدلی که ما ساختیم، باعث تضعیف مدل شد. البته اگر مدل را خیلی بزرگتر میکردیم، جواب میداد، ولی مشکلی که با colab داشتم و اینکه آموزش مدل خیلی طول میکشید، باعث شد خیلی نتونم همه حالات رو بررسی کنم. تقریبا ۶ ساعت از زمانم روی آموزش مدلهای مختلف رفت.

حالت dropout, data augmentation, regularization:

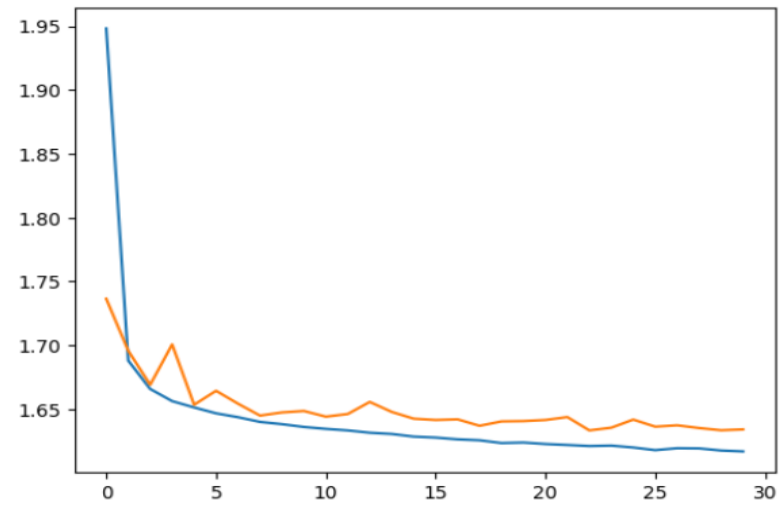
نمودار تابع ضرر:



مقدار accuracy:

```
Test Accuracy: 0.8285
Train Accuracy: 0.8493166666666667
```

حالت regularization, data augmentation:

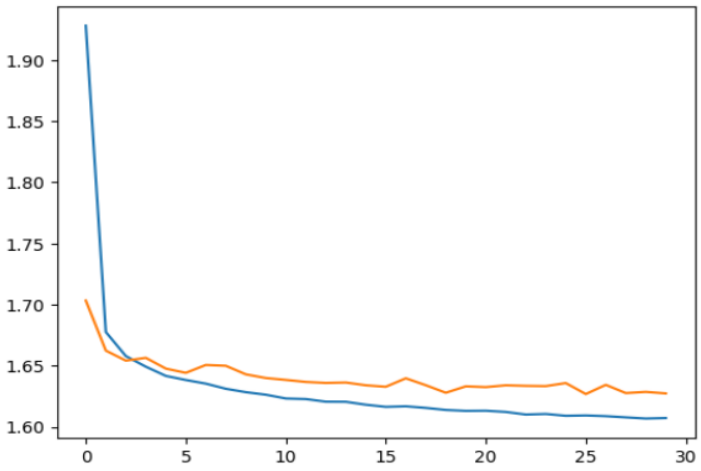


مقدار accuracy:

```
Test Accuracy: 0.8282
Train Accuracy: 0.8462166666666666
```

حالت regularization, dropout:

نمودار loss:



مقدار accuracy:

```
Test Accuracy: 0.8332
Train Accuracy: 0.8548666666666667
```

متاسفانه هر ترکیبی رو امتحان کردم، بیشتر باعث بدتر شدن میشد تا بهتر شدن و حدسم اینه که مدل رو خیلی داریم محدود میکنیم و مدل اولیه‌مون که برای بخش ب زدیم، اونقدر ظرفیت بالایی نداشته و overfit نشده که حالا با این روشها بخواد بهتر بشه. پیچیده‌تر کردن مدل زمان آموزش خیلی بیشتری می‌طلبید و همین مدلی که برای بخش ب زدم، برای ۶۰ epoch حدود ۵۰ دقیقه زمان برد.

به نظرم بهترین ترکیب مربوط به استفاده از regularization به تنهایی بود که باعث شد دقت مدل روی تست و آموزش بالا بره.