بسم الله الرحمن الرحیم

محمد عرفان زارع زردینی

۹۸۴۱۱۴۳۲

تمرین سری یک درس یادگیری عمیق

مدرس درس: استاد داوود آبادی

پاییز 1402

1)

منابع:

الف)

https://medium.com/mlearning-ai/overfitting-vs-underfitting-6a41b3c6a9ad#:~:text=Overfitting%20occurs%20when%20the%20model%20is%20complex%20and%20fits%20the,find%20relationships%20and%20patterns%20accurately.

https://docs.aws.amazon.com/machine-learning/latest/dg/model-fit-underfitting-vs-overfitting.html

https://www.geeksforgeeks.org/underfitting-and-overfitting-in-machine-learning/

ب)

Chatgpt(prompt: how can I find overfitting in trained model?)

پ)

پاسخ ها:

الف)

در شبکه عصبی بیش برازش زمانی رخ می دهد که مدل بیش از حد پیچیده بشود و نویز و نوسانات تصادفی ایجادی در دیتا های آموزشی را یاد بگیرد ، که سبب تعمیم ضعیف(poor generalization) در داده هایی می شود که دیده نشده اند. بیش برازش با خطای آموزش کم و

خطای اعتبارسنجی بالا، پیچیدگی بیش از حد و ناتوانی مدل در عملکرد خوب در داده های جدیدمان، مشخص می شود. برای کاهش بیش برازش، روش هایی همچنین generalization، dropout، متوقف کردن زودتر از موعد مقرر و استفاده از داده های آموزشی بیشتر مناسب است. کم برازش زمانی اتفاق می افتد که مدل بیش از حد ساده باشد ونتواند الگو های اولیه و پایه ای را در داده ها بیاموزد. هم چنین با خطای آموزشی و اعتبار سنجی بالا و پیچیدگی ناکافی ،مشخص می شود. برای اصلاح این مورد میتوان پیچیدگی مدل را زیاد نمود و کیفیت و تعداد داده های آموزش را بهبود بخشید و هایپر پارامترهارا تنظیم نمود.

برای رسیدن به generalization مناسب ، باید تعادلی میان بیش برازش و کم برازش برقرار نمود. باید مدل پیچیدگی مناسبی داشته باشد تا الگو های مورد نیاز را بخوبی و بدون حساسیت اضافی به نویز و موارد اضافه یا موارد ساده و پایه، یاد گیرد.

ب)

برای آنکه بررسی نماییم که مدل از پیش آموزش دیده ما ، دچار بیش برازش شده است یا نه نکات زیر را بررسی می نماییم:

1-داده های به سه قسمت آموزش ، اعتبارسنجی و تست تقسیم نموده (hold-out) و بررسی مینماییم. اگر عملکرد مدل در مجموعه تست بسیار بدتر از داده های آموزشی باشد، نمایان کننده بیش برازش است.

2- خطا و ضرر را روی یک داده اعتبارسنجی محاسبه نموده، اگر خطای محاسبه شده ، به شکل زیادی بیشتر از خطای آموزش باشد، از نشانه های بیش برازش است.

3-منحنی آموزش(learning curves) را برای نمایش دادن خطاهای آموزش و اعتبارسنجی رسم نموده، اگر خطای اعتبارسنجی زیاد بماند یا حتی افزایش یابد ، درحالیکه خطای آموزش کم شده ، نمایان کننده ی بیش برازش می باشد.

4- از روش cross-validation استفاده نموده و مدل را روی زیرمجموعه های مختلف داده ها آموزش داده و عملکرد آن را روی باقی داده ها ارزیابی می نماییم. اگر مدل روی داده های آموزش بسیار بهتر از داد های اعتبارسنجی بود، بیش برازش رخ داده.

پ)

از dropout برای جلوگیری از بیش برازش استفاده می شود . این روش تکنیکی منظم ساز در جهت جلوگیری از overfitting در شبکه عصبی با صفر نمودن (به صورت تصادفی اغلب) کسری از واحد های ورودی در هر مرحله آموزش می باشد که به وابستگی بیش از حد شبکه به ورودی های خاص کمک می نماید و آن را تشویق می کند تا ویژگی های قوی تری را آموزش ببینند.

در این سوال باتوجه به مشخص شدن dropoutmask در جدول دوم، فقط لازم است تا در عنصر نظیر آن در جدول اول ضرب شود تا مقادیر نهایی آن مشخص شود که در جدول سوم، نتیجه را نوشته ام. همچنین برای مقادیرنهایی در مرحله آزمون ، از باید ابتدا مقدار p را با توجه به عدم مشخص شدن آن توسط سوال ، محاسبه نمایید که فرمول آن بدین شکل است که تعداد صفر های موجود را بر تعداد کل داده ها تقسیم می نماییم. حال تنها کافی است هر عنصر از جدول 1 را در (1-p) ضرب نماییم تا خروجی حاصل شود که در جدول چهارم قابل نمایش است.

جدول ۱: *Output*

۱.۹	-۰.۲	-۰.۷	۱.۶
-۰.۹	۲.۵	۲.۵	-۲.۳
-۰.۴	۳.۷	۳.۲	-۰.۵
۱.۲	-۲.۶	-۰.۴	۱.۳

جدول ۲: *Dropoutmask*

۱	۰	۰	۱
۰	۱	۱	۰
۰	۱	۱	۰
۱	۰	۰	۱

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

جدول سوم

مقدار pبرابر است با:

P=8/16=0.5

1-p=1-0.5=0.5

بدین سان خروجی حالت تست می شود:

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

جدول چهارم

2)

الف)

منبع:

https://medium.com/swlh/k-nearest-neighbor-ca2593d7a3c4

ghatgpt(prompt: in knn algorithm, when we change parameters k, bias and variance change. But how ? they goes up or not? Explained it.)

یک الگوریتم در حوزه supervised learning است که برای کارهای رگرسیون و طبقه بندی استفاده می شود. بدین گونه است که کلاس یا مقدار یک نقطه داده حدید را براساس شباهت به

نقاط آموزشی مجاور ، پیش بینی می نماید.بدین سان این الگوریتم داده های تست متعلق به هر کلاس را به کلاسی با بیشترین احتمال اختصاص می دهد. پیرامون بایاس و واریانس، مقدار کم k سبب می شود تا مقدار بایاس و واریانس آن بسیار بالا باشد ، زیرا مدل به طور زیادی به نزدیک ترین همسایه واسبته می شود که می تواند سبب بیش برازش شود. این بدین معنی است که مدلمان الگو های پایه در داده را ثبت نمی نماید و در داده های جدید، ضعیف عمل می نماید.همچنین با مقدار k کم، مرز تصمیم(decision boundary) نسبت به نقاط داده فردی در مجموعه آموزشی انعطاف پذیرتر و حساستر می شود. بدین سان تغییرات کوچک در داده آموزش، سبب تغییر زیاد بر نتیجه پیش بینی می شود.چ که سبب تاثیر نویز و نقاط پرت در داده های آموزشی می شدو که سبب افزایش واریانس شده است.در واقع مدلمان بیش از حد داده های آموزشی را برازش می کند که سبب می شود داده های آموزش و برچسبشان را بخاطر بسپارد(که این روش اشتباهی است) . در نتیجه مدل ممکن است که به خوبی به داده های جدید . نادیده تعمیم ندهد و درنتیجه واریانس بالایی منجر می شود.

همچین در مقدار k زیاد منجر به بایاس پایین و واریانس بالا می شود. مرز تصمیم smooth تر و تعمیم یافته تر شده(بر داده های جدید) که سبب شود تاثیر نقاط آموزشی کاهش یابد و سبب generalization بهتر می شود.

نکته ای که وجود دارد این است که اگر k خیلی بزرگ باشد، میتواند سبب بایاس بالا شود . مدل داده ها را بیش از حد ساده می نماید و سبب عدم تناسب میانشان می شود. این حالت زمانی رخ می دهد که مدل برای ثبت پیچیدگی های پایه داده ها ،بسیار ساده عمل نماید.

ب)

استفاده از منظم‌سازی، ممکن است باعث تضعیف عملکرد مدل شود.

پاسخ:

درست. منظم سازی روشی برای جلوگیری از بیش برازش است که با اضافه نمودن یک جریمه(penalty) به تابع مدل استفاده می شود. این جریمه مدل را از وابستگی بیش از حد به

فیچر ها یا پیچیده شدن بیش از حد باز می دارد و بدین سان مدل را پیرامون بیش برازش کنترل می نماید. البته روی مدل و عملکردش تاثیر می گذارد و انعطاف پذیری مدل را کم می نماید و بدین سان اگر از منظم سازی استفاده نشود با داده های آموزشی تطابق نداشته باشد که این مورد سبب کاهش عملکرد مدل می شود. البته منظم سازی، به مدل کمک می کند به داده های دیده نشده تعمیم بهتری داشته باشد و از overfit شدن جلوگیری نماید که اغلب در عملکرد مطلوب تر است.

اضافه کردن تعداد زیاد ویژگی‌های[۴] جدید، باعث جلوگیری از بیش‌برازش می‌شود.

پاسخ:

غلط.لزوما افزودن تعداد زیادی ویژگی از بیش برازش جلوگیری نمی نماید و حتی میتواند احتمال آنرا نیز افزایش دهد.افزودن فیچر ، سبب ایجاد فضای ویژگی با ابعاد بالا شود و مدل را به سمت overfitting سوق دهد. باید فیچر ها بهه درستی انتخاب شوند و موارد غیر ضروری یا اضافی انتخاب نشوند. بیش برازش زمانی رخ می دهد که مدل بیش از حد پیچیده شده و به جای یادگیری الگوهای اساسی، شروع به ثبت نویز و نوسانات تصادفی در داده های آموزشی نماید.روش های منظم سازی و انتخاب ویژگی، می تواند سبب کاهش خطر بیش برازش با انتخاب آموزنده ترین ویژگی ها و کاهش تاثیر موارد نامربوط بکار رود.

با زیاد کردن ضریب منظم‌سازی، احتمال بیش‌برازش بیشتر می‌شود.

پاسخ:

غلط. ما از این روش برای کاهش احتمال بیش برازش استفاده می نامییم و افزایش ضریب آن نیز احتمال بیش برازش را کاهش می دهد. ضریب منظم سازی بزرگتر، درواقع penalty ضرایب پیچیده را بیشتر می نماید و مدل را به سمت سادگی می برد. بدین سان از بیش برازش جلوگیری می نماید. با افزایش این ضریب، مدل محدود شده تا مقادیر پارامتر کوچکتری را بکارگیرد. این

عامل با محدود کردن پیچیدگی مدل به جلوگیری از بیش برازش کمک می کند.پس بدین سان با موضوع سوال در تناقض است و متن سوال غلط است.

پ)

برای آنکه متوجه شویم که از کدام تکنینک منظم سازی استفاده شده است، باید وزن های بدست آمده در هر آزمایش را تجزیه و تحلیل کنیم.

آزمایش اول:

$$W_{exp1} = [0.26, 0.25, 0.25, 0.25]$$

وزن ها نستا کوچک و از نظر رنج و تفاوت مشابه اند و دریک رنج اند. پس گویی نوعی از منظم سازی رو آن اعمال شده است. از آنجا که وزن ها به گونه ای نیست که تعدادی صفر شده باشند ، نمایانگر منظم سازی L1 است . این روش منظم سازی با افزودن مقادیر ثابت وزن ها به تابع ضرر،پراکندگی را پرورش می دهد. پس طبف این ویژگیها به نظر منظم سازی L1 است.

آزمایش دوم:

$$Wexp2 = [1, 0, 0, 0]$$

وزن ها به جز یکی از آنها، صفر اند که نمایانگر نوعی منظم سازی برای پرورش پراکنندگی و افزایش تعداد وزن های غیر صفر هست(با کاهش وزن ها به سمت صفر). در منظم سازی نوع L2 ، ما مقادیر وزن هارا مجذور نموده و به تابع ضرر می افزاییم. از انجایی که در اینجا یک وزن غیرصفر است،گویی این نوع منظم سازی اعمال شده . زیرا این نوع منظم سازی، تمایل دارد وزن ها را به سمت صفر کاهش دهد که البته سبب پراکندگی دقیق نمی شود. پس با توجه به شواهد به نظر ارر منظم سازی L2 استفاده شده است.

آزمایش سوم:

$$W_{exp3} = [13.3, 23.5, 53.2, 5.1]$$

نکته مشهود در این آزمایش ، وزن های بزرگ و بالا آن و تفاوت در رنج و محدوده های آن است.هیچ الگوی مشهودی از پراکندگی یا وزن صفر وجود ندارد. چیزی که به نظر می رسد این است که هیچ منظم سازی واضحی در آزمایش اعمال نشده است. با توجه به آنکه هدف روش های منظم سازی همچون L1,L2 کوچک نمودن وزن ها و پروراندن پراکندگی است و در این نمونه داده ، وزن ها ویژگی های گفته شده را ندارند، پس گویی از منظم سازی در این آزمایش استفاده نشده بوده است.

آزمایش چهارم:

$$W_{exp4} = [0.5, 1.2, 8.5, 0]$$

وزن ها دارای محدوده و رنج متفاوتی اند و وزن صفر داریم. پس نوعی منظم سازی اعمال شده است. هردو منظم سازی مدنظرمان میتوانند پروراندن پراکندگی آن باشند ولی در L1 ما میخواهیم با رساندن برخی از وزن هایمان به صفر ، پراکنندگی را داشته باشد. در این آزمایش ، وجود وزن صفر نمایانگر این است که گویی از L1 استفاده شده است.

3)

الف)

مرجع:

https://neptune.ai/blog/knowledge-distillation

https://blog.roboflow.com/what-is-knowledge-distillation/#:~:text=Knowledge%20distillation%20is%20a%20powerful,language%20processing%2C%20and%20speech%20recognition.

تقطیر دانش، فرآیندی برای انتقال دانش از مدل پیچیده معلم به یک مدل دانش اموز که ساده تر است می باشد. هدف، بهبود عملکرد مدل دانش آموز با استفاده از اطلاعات مدل معلم که پیچیده تر است، می باشد. در این فرآیند، مدل معلم، مدلی بزرگتر و دقیق همچون شبکه عصبی عمیق است که روی مجموعه داده بزرگ آموزش داده شده است. درحالی که مدل دانش آموز کوچکتر و فشرده تر است و هدفی همچون تکرار عملکرد مدل معلم با محاسباتی کمتر است. در واقع انتقال دانش به دانش آموز برای تقلید از رفتار خروجی مدل معلم است. در واقع مدل دانش اموز در آموزش، از نتایج و soft target خروجی های مدل معلم یاد می گیرد که در واقع soft target اطلاعات دقیقی را درباره اطمینان نسبی و شباهت میان کلاس های مختلف میدهد. در آموزش مدل دانش آموز تلاش میکند تا اختلاف نتایج را با نتایج مدل معلم کم نماید. این کار ها سبب می شود تا مدل دانش اموز جزئیات دقیقی را بررسی کند و با معماری کمتر تعمیم دهی بهتری داشته باشد. هدف از تقطیر دانشْ رسیدن به فشرده سازی مدل و بهبود کارایی و استقرار مدلمان است. این روش با انتقال دانش معلم به دانش اموز (مدل بزرگتر به کوچکتر) امکان ایجاد مدل فشرده تری را حاصل می سازد و سبب می شود تا مدلمان به عنوان جایگزینی سریعتر وکارآمد تر برای مدلی پیچیده استفاده شود.

ب)

منبع:

Chatgpt(prompt: whats hard prediction and soft prediction)

(در این سوال برای راحتی در تایپ ، برای بخشی از قسمت ها از معادل عینی فارسی استفاده شده است.)

معماری آن نوعی از تقطیر دانش است و انتقال دانش معلم به دانش اموز می باشد. فرآیند یادگیری با داده های ورودی شروع می شود که به دو مدل داده می شود. در مدل معلم پس از گذشتن از m لایه ، خروجی وارد تابع softmax می شود که خروجی آن نشان دهنده برچسب های نرم است که حاوی اطلاعات ظریفتری پیرامون اطمینان نسبی و شباهت میان کلاس های

مختلف است. در مدل دانش آموز داده های ورودی از N لایه عبور داده شده و وارد دو تابع softmax می شود که خروجی softmax(T=t) پیش بینی خود مدل مان است . پیش بینی نرم با برچسب هایی که مدل معلم به دست آورده (برچسب نرم) مقایسه شده و تفاوت بین پیش بینی شان محاسبه میش ود در تابع ضرر. در خروجی softmax(t=1) در مدل دانشجویی که پیش بینی سخت در نظر گرفته می شود، با برچسب اصلی و درست مقایسه می شود و هدف در اینجا، اطمینان حاصل می شود که پیش بینی های سخت مدل دانش آموز با truth ground همسو شود. تابع ضرر در اینجا برای تعیین اختلاف میان پیش بینی نرم و برچسب نرم ، و اختلاف میان پیش بینی سخت و لیبل اصلی است. فرآیند یادگیری پیرامون به حداقل رساند تابع ضرر می باشد که با backpropagation و گزادیان نزولی انجام می شود که البته گرادیان با توجه به پارامتر های هر دو مدل محاسبه می شود و سبب بهینه سازی هردو مدل می شود. با بهینه سازی تابع ضرر، مدل دانش اموز یادمیگیرد که پیش بینی های نرم مدل معلم را تقلید نمایدو دانش و اطلاعات منتقل شده توسط برچسب های نرم را بدست آورد. توجه شود که پیش بینی سخت به برچسب های کلاس گسسته اشاره دارد که توسط مدل دانش آموز با تابع softmax(t=1) اختصاص دارد. پیش بینی نرم به توزیع پیوسته احتمالات کلاس اشاره دارد که تابع softmax(T=t) اعمال می شود. پیش بینی نرم، تخمین دقیق تر و پیوسته تری را از احتمالات کلاس ارائه می دهد. درواقع تابع softmax(T=1) خروجی مدل را به توزیع احتمال در کلاس های مختلف تبدیل نموده و کلاس پیش بینی شده، اغلب کلاسی ا بالاترین امتیاز هست و با برچسب کلاس مطابقت دارد.در حالی که پیشبینی نرم، با SOFTMAX(T=t) احتمالات را پراکنده می نماید و احتمالات غیرصفر را به چندین کلاس اختصاص میدهد که نشان دهنده عدم قطعیت و شباهت مدل میان کلاس هایمان بود. این پیش بنی تخمینی دقیق و پیوسته ای از احتمالات کلاس را نسبت به پیش بینی سخت دارد. پیش بینی نرم حاوی اطلاعات بیشتری نسبت به دگیری اسست و در فرایند انتقال دانش از آن استفاده میشود.

ج)

تابع ضرر در دو قسمت استفاده شده(با توجه به تصویر بخش قبلی) دربخش بالایی، ضرر با مقایسه توزیع احتمال میان پیش بینی نرم و برچسب نرم، با استفاده از cross-entropy

محاسبه می شود. در تابع ضرر پایینی که تفاوت برچسب اصلی و پیش بینی سخت دانش اموز است که برای کمک بهتر پیش بینی نمودن و پیش بینی دقیق نمودن است. تابع ضرر مناسب آن نیز میتوان به mean squaredو cross-entropy اشاره نمود. همچنین از یک تابع ضرر مبتنی بر گرادیان نیز همچون sgd برای بروز رسانی استفاده می شود.تا جهت و میزان به روز رسانی وزن ها برای به حداقل رساندن تلفات و بهبود عملکرد دانش آموز را نشان دهد.

(4

ابتدا کتابخانه های مورد نیاز را اضافه می نمایید.

در سلول بعدی، کلاس model را تعریف نموده ایم که شامل تابع تعریف اولیه آن برای مقدار دهی متغیر های ورودی و لیبل ها می باشد و فراخوانی توابع نوشته شده روی آن ها می باشد. تابع initialize_moms_ ، ممنتم دو وزن و بایاس را مقدار دهی اولیه نموده ایم که صفر اند در ابتدا. در تابع initalize_RMSs _ مقدار rms(میزان دقت وخطای پیش بنی را محاسبه میکند) دو وزن و بایاس را مقدا دهی اولیه صفر می نماید. تابع random_tensor_ نیز بدین سان است که تنسوری با اندازه مشخص شده است که یک تنسور با عناصر نمونه برداری از توزیع نرمال استاندارد با میانگین صفر و واریانس ایجاد می کند. همچنین requires_grad_ می آید که گرادیان تنسور باید در حین انتشار برای تمایز خودکار محاسبه شود که این ویژگی است که فعال می شود و گرادیان ها را ردیابی نمی کند. در تابع initalize_parameters_ می آید پارامتر هایمان را با استفاده از تابع random_tensor_ مقدار دهی می نماید که تنسور است با سایزی که مشخص شده است. برای وزن لایه1 به اندازه بعد دوم شکل ورودی(x) که همان تعداد ستون ها می باشد و بعد دیگر آن هم 3 است . بایاس آن هم با سایز 1 می باشد. همچنین وزن لایه 2 سایزی 1*3 دارد. و بایاس آن هم طبیعتا طبق دانشمان باید سایز 1 باشد. تابع loss_func_ نیز می آید تفاوت مقدار پیش بینی و اصلی را محاسبه می کند و به توان دو میرساند و میانگین میگیرد که همان mean squared است. در تابع nn_ هم ورودی xb که یک تنسور است را ، تبدیل خطی روی آن میزند و می آید و ضرب ماتریسی میان وزن لایه اول و

تنسور ورودی انجام داده و بایاس را می افزاید .سپس در خط بعد،حداکثر مقدار میان l1 و تنسور 0.0 میگیرد. عملیات بیان شده میتوان نمایانگر تابع فعالساز relu داسنت که غیرخطی بودن را به شبکه عصبی میبرد. در خط بعد هم l3 را به شکل ضرب نقطه ای میان l2(خروجی لایه قبلی) و وزن لایه دوم بوده و بایاس لایه دوم به آن اضافه می شود و return می شود .

در تابع train با ورودی بهینه ساز مشخص شده و تعریف شده، ابتدا مقدار نرخ آموزش های مختلف تعریف شده و سپس بر اساس آن ها امده ، تا زمانی طول لیست خطا 0 یا آخرین خطای اضافه شده بزرگتر از 0.1 باشد وهمچنین تعداد خطاها از 1000 تا کمتر است، محاسبات لایه هاا را انجام داده و تابع ضرر را محاسبه می نماید. سپس یک backward روی خطای محاسبه اعمال می شود. سپس ورودی های تابع های بهینه ساز را مشخص نموده و فراخوانی می نماییم.(برای هر لایه تابع بهینه ساز را فرا می خوانیم. و در نهایت به لیست خطاها اضافه مینماییم. برای نمایش هم با توجه به چند نمودار بودن ابتدا جایگاه آن را مشخص و سپس بازه نمایش داده هارا میان 0تا 30 مقادیر خروجی را رسم می نماییم. ودر خط کد torch.manual_seed برای تولید اعداد تصادفی در بخش های بعدی، هر بار که کد با آن seed اجرا شود ، دنباله اعداد تصادفی یکسانی را تولید می کند. که میتواند برای تکرارپذیری خوب باشد و در مقایسه نتایج و اشکال زدایی کمک نماید. درنهایت هم در رسم ابتدا هر روی لیست نمودار ها پیشروی نموده، ولیست را به فرمت یک بعدی تبدیل می نماید .و در خط plt.tight_layout() می آید از هم پوشانی وشلوغی نمودار وبرچسب ها عناوین وعناصر دیگر جلوگیری می نامید وبه صورت خودکار چیدمان نمودار ها را برای بهینه سازی فاصله تنظیم می کند.

```python
class model:
    def __init__(self, x, y):
        self.x = x
        self.y = y

        self._initalize_parameters()
        self._initalize_moms()
        self._initalize_RMSs()

    def _initalize_parameters(self):
        self.weights_1 = self._random_tensor((x.shape[1],3))
        self.bias_1 = self._random_tensor(1)
        self.weights_2 = self._random_tensor((3,1))
        self.bias_2 = self._random_tensor(1)

    def _random_tensor(self, size): return (torch.randn(size)).requires_grad_()

    def _initalize_moms(self):
        self.moms_w1, self.moms_b1 = [0], [0]
        self.moms_w2, self.moms_b2 = [0], [0]

    def _initalize_RMSs(self):
        self.RMSs_w1, self.RMSs_b1 = [0], [0]
        self.RMSs_w2, self.RMSs_b2 = [0], [0]
    def _nn(self, xb):
        l1 = xb @ self.weights_1 + self.bias_1
        l2 = l1.max(torch.tensor(0.0))
        l3 = l2 @ self.weights_2 + self.bias_2
        return l3

    def _loss_func(self, preds, yb):
        return ((preds-yb)**2).mean()
```

```python
def train(self, optimizer):
    # Multiple learning rates to see how optimizers work with them
    lrs = [10E-4,10E-3,10E-2,10E-1]
    ## for plotting ##
    fig, axs = plt.subplots(2,2)
    ## for plotting ##
    all_losses = []
    for i, lr in enumerate(lrs):
        losses = []
        while(len(losses) == 0 or losses[-1] > 0.1 and len(losses) < 1000):
            preds = self._nn(self.x)
            loss = self._loss_func(preds, self.y)
            loss.backward()
            optimizer(self.weights_1, lr, self.moms_w1, self.RMSs_w1)
            optimizer(self.bias_1, lr, self.moms_b1, self.RMSs_b1)
            optimizer(self.weights_2, lr, self.moms_w2, self.RMSs_w2)
            optimizer(self.bias_2, lr, self.moms_b2, self.RMSs_b2)
            losses.append(loss.item())
        all_losses.append(losses)

        ## for plotting ##
        xi = i%2
        yi = int(i/2)
        axs[xi,yi].plot(list(range(len(losses))), losses)
        axs[xi,yi].set_ylim(0, 30)
        axs[xi,yi].set_title('Learing Rate: '+str(lr))
        ## for plotting ##

        # Setting seed makes sure the parameters are initalized the same way for better comparison
        torch.manual_seed(42)
        self._initalize_parameters()
        self._initalize_moms()
        self._initalize_RMSs()

    ## for plotting ##
    for ax in axs.flat:
        ax.set(xlabel='steps', ylabel='loss (MSE)')
    plt.tight_layout()
    ## for plotting ##
```

در سلول بعد تابع generate_fake_labels قرار دارد که براساس ورودی های هر بخش تنسور داده ورودی، با ضرایب مشخص شده عددی را خروجی میدهد. سلول بعدی هم ورودی را تعریف و خروجی را به شکل تنسوری با سایز 5 و با استفاده از تابع قبلی تعریف شده است. سپس متغیری برای کلاس model با استفاده از ورودی و خروجی تعریف می نماییم.

```
def generate_fake_labels(x3, x2, x1):
    return (x3**3 * 0.8) + (x2**2 * 0.1) + (x1 * 0.5) + 4.
```
✓ 0.0s

```
x = torch.tensor([[0.7,0.3,0.7],
                  [0.4, 1., 0.4],
                  [0.2, 1.1, 0.1],
                  [0.4, 0.7, 0.2],
                  [0.1, 0.5, 0.3]])
y = torch.tensor([generate_fake_labels(i[0],i[1],i[2]) for i in x])
print(x.shape, y.shape, y)
```
✓ 0.0s

```
torch.Size([5, 3]) torch.Size([5]) tensor([4.6334, 4.3512, 4.1774, 4.2002, 4.1758])
```

```
y = torch.tensor([4.6334, 4.3512, 4.1774, 4.2002, 4.1758])
```
✓ 0.0s

```
my_model = model(x, y)
```
✓ 0.0s

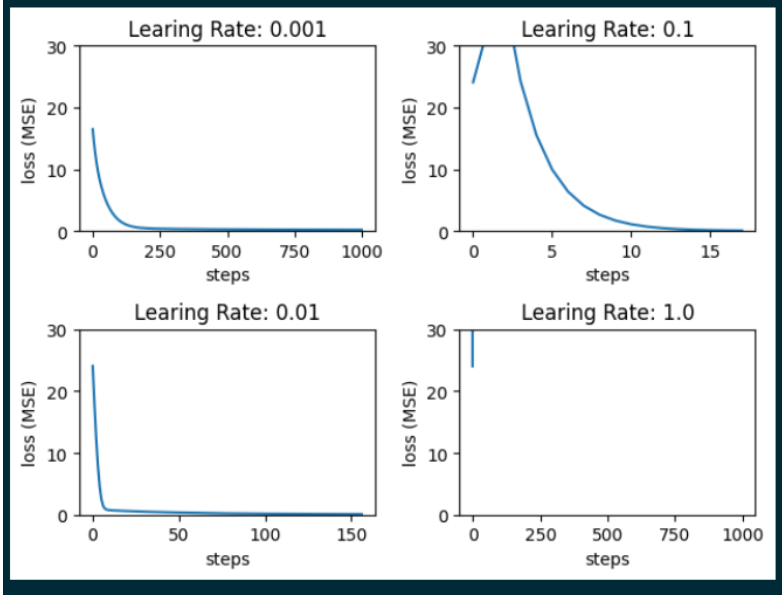
در ادامه دو بهینه ساز ممنتم وsgd تعریف نموده شده و به تابع آموزش مدلمان دادیم و خروجی آن که نمودار های آن براساس نرخ آموزش است را نمایش داده است.در تابع SGD ، مقدار وزن ورودی را براساس گرادیان آن و نرخ آموزش بروز می نماییم. در خط بعدی هم گرادیان را none کردیم که در مراحل بعد دوباره استفاده شود . در تابع momentum هم ابتدا مقدار ممنتم وزن در اخرین مرحله را ذخیره نموده و ممنتم جدید را با استفاده از مقدار گرادیان وزن ورودی و نرخ آموزش و ممنتم اخر، محاسبه می نماییم که فرمول آن در کد موجود است . سپس مقدار را به لیست ممنتم اضافه نموده و از دیتامی کاهیم و گرادیان انرا هم none می نماییم تاد در جلوتر استفاده شود.(نتایج تحلیل نمودار ها پس از شکل نوشته شده است)

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

✓ 0.0s

```
#train model with SGD
my_model.train(SGD)
```

✓ 1.2s

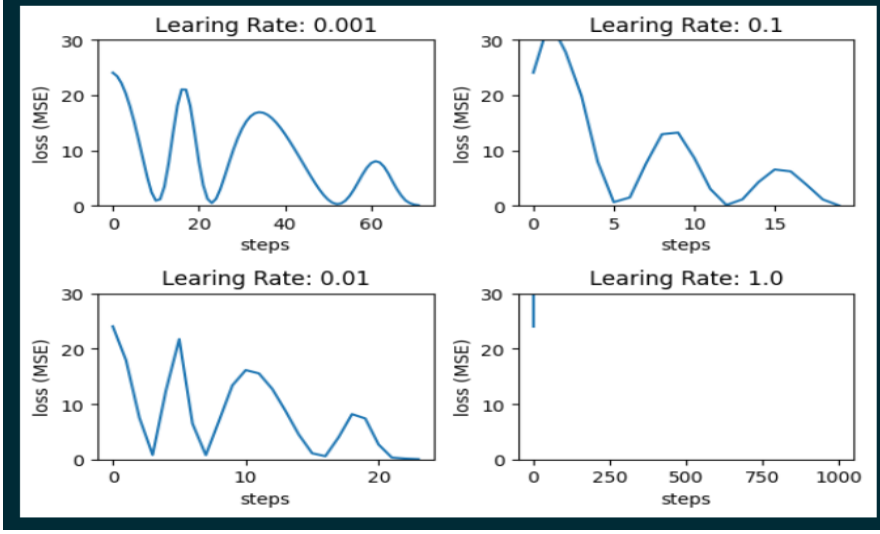



```
def momentum(a, lr, moms, __):
    previous_momentum = moms[-1]

    mom = a.grad * lr + previous_momentum * (1-lr)
    moms.append(mom)
    a.data -= mom
    a.grad = None
```

✓ 0.0s

```
#train model with momentum
my_model.train(momentum)
```

✓ 0.7s



در بهینه ساز ممنتم، ما مشاهدهد میکنیم که داده ها با نرخ اموزش 0.001 ، خروجی تابع ضرر آن به شکل سینوسی و نوسانی است و در نهایت در حوالی گام 70 به صفر میل می کند. درحالی که با افزایش نرخ یادگیری به 0.01 با سرعت بیشتری به صفر میل میکند و با افزایش بیشتر آن به 0.1 مشاهده می شود که خطا سریعتر میل می نماید ولی در ابتدا خطا با آنکه افزایش میابد ولی به مرور کم وکمتر می شود و قله آن کاهش یافته و به صفر میل می نماید .این درحالی است که افزایش بیش از حد نرخ یادگیری سبب می شود که مدلمان خروجی با خطای بالا داشته باشد و تابع ضرر بدرستی عمل نکند.پیرامون گرادیان کاهشی هم همین روند رایج و نمایان است ولی با این نکته که به شکل لگاریتمی کاهش یافته و روند نزلی در 0.01و0.001 دارد و در 0.1 هم با انکه روند آن ابتدا در خروجی ضرر صعودی بوده ولی به سرعت روند اصلاح ودرست شده و در گام های سریعتری به صفر متمایل شده است.در 1 هم همانند مشکل ممنتم تابع ضرر عملکردش رررا به درستی نمی تواند اجرا نماید و به مشکل می خورد.

5)

الف)

ابتدا بخش های اولیه را ران می نماییم تا کتابخانه ها و موارد موردنیاز اضافه شود و همچنین در سلول های بعد و یک ترنسفورم برای نرمالایز کردن داده ها و همچنین داده های آموزش و تست دانلود و پردازش شود و در نهایت سلول دیگری که برای نمایش داده های دیتاست مان بود را ران مینماییم. حال در سلول بعدی مدل مدنظرمان را که مدل خطی شامل یک لایه تخت و 4 لایه اصلی به همراه فعالساز ReLU و تابع فعالساز نهایی نیز، logsoftmax زدم که یک dim را به یک تقلیل میدهد. مدل براساس سایز ورودی و خروجی که در سلول قبل داده شده است تنظیم شده است. همچنین با استفاده از کتابخانه torchو بخش های torch.optim وtorch.nn تابع ضرر crossentropy و بهینه ساز sgd با نرخ یادگیری 0.01(که براساس مشاهدات و تست انواع مختلف حاصل شد) تعریف نمودیم. همچنین مدل را برای مشاهد چاپ نمودیم .در قسمت تابع آموزش نیز برای بخش forward pass تصویر را در هر حلقه به مدل دادیم.

همچنین برای خطا نیز خروجی حاصل از بخش قبل و لیبل هارا بدان میدهیم. با ران نمودن سلول خطای آموزش را در epoch =10 مشاهده می نماییم. در نهایت نیز کتابخانه d2l را افزوده بر روی داده های تست بررسی می نماییم و خروجی را با لیبل اصلی و تصویر آن کنار هم چاپ مینماییم. همانطور که میبینید لیبل پیش بینی شده و اصلی منطبق اند.

تصاویر کد ها و خروجی ها نیز در پایین موجود است.

```
## Train your model
epochs = 10

for e in range(epochs):
  running_loss = 0
  for images, labels in trainloader:

    images = images.view(images.shape[0],-1)

    #reset the default gradients
    optimizer.zero_grad()

    # forward pass
    ################ Your code ############
    output = model(images)
    loss =criterion(output,labels)
    #######################################

    loss.backward()
    optimizer.step()

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

```
Training loss: 0.16336006829654104
Training loss: 0.15311268383839619
Training loss: 0.14895502458820972
Training loss: 0.14496693829856894
Training loss: 0.14186166745545006
Training loss: 0.13860499760362385
Training loss: 0.13563115255776118
Training loss: 0.13240618269239215
Training loss: 0.12910449424031764
Training loss: 0.12648769992508932
```

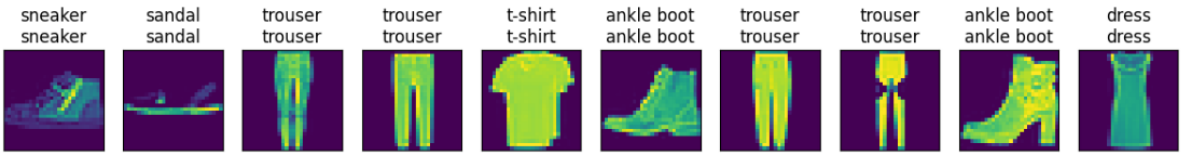
```
[5] input_size = 784
    out_size = 10
```

```
## Define the model
################ Your code ############
model = nn.Sequential(
    nn.Flatten(),
    nn.Linear(784,512),
    nn.ReLU(),
    nn.Linear(512,256),
    nn.ReLU(),
    nn.Linear(256,128),
    nn.ReLU(),
    nn.Linear(128,10),
    nn.LogSoftmax(dim=1)
)
#######################################
```

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

```
[8] print(model)
```

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



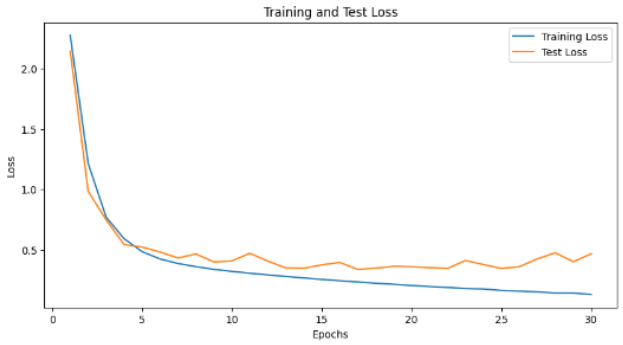
ب)

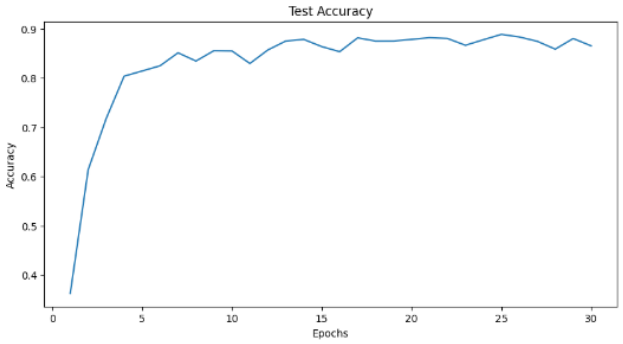
مدل تغییر یافته مان دچار بیش برازش شد.بدین سان که تعداد لایه ها را کمی زیاد کردم و همچنین تعدا نورن های آن نیز تغییر داده شد.همچنین تعداد epoch نیز برای بررسی بهتر ، افزایش یافت. دلایل بیش برازش شدن هم برمی گردد به افزایش پیچیدگی در مدل نسبت به اندازه مجموعه داده آموزشی مان که سبب می شود که بجای یادگیری ، گویی حفظ کندکه سبب عملکرد ضعیف در داده های دیده نشده می شود که البته بخشی از آن برمی گردد به اموزش زیاد مدل به علت افزایش epoch . البته در کل بیش برازش دلایل دیگری همچون داده آموزشی محدود و پارامترهای آموزشی و عدم استفاده از منظم سازی نیز می تواند داشته باشد که اینجا مورد بحث و مدنظر به عنوان دلیل نیست. همچنین همانطور که از نمودار و داده ها مشخص است، ضرر در آموزش به شکل منظمی کاهش میابد ولی در تست ابتدا کاهش وسپس افزایش یافته که نشان از بهبود در عملکرد خود در مدل است ولی پس از چند دوره، ضرر آن دوباره زیاد شده است که نشان از بیش برازش است و مدل برروی داده های دیده نشده تعمیم نیافته است.
همچنین دقت داده تست پس از بهبود اولیه ثابت مانده است که نشان میدهد که مدل فراتر از الگوهای موجود در داده های آموزشی یاد نمی گیرد.(در دقت و عملکردی ضعیف روی تست داشته است.) همچنین همانطور که مشخص است در epoch های انتهایی تفاوت بین خطا افزایش فاحشی داشته است.

```
model = nn.Sequential(
    nn.Flatten(),
    nn.Linear(784, 1024),
    nn.ReLU(),
    nn.Linear(1024, 2048),
    nn.ReLU(),
    nn.Linear(2048, 2048),
    nn.ReLU(),
    nn.Linear(2048,1024),
    nn.ReLU(),
    nn.Linear(1024, 256),
    nn.ReLU(),
    nn.Linear(256, 128),
    nn.ReLU(),
    nn.Linear(128, 10),
    nn.LogSoftmax(dim=1)
)
```

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

```
epochs = 30

train_losses = []
test_losses = []
test_accuracies = []

for e in range(epochs):
    running_loss = 0
    for images, labels in trainloader:
        images = images.view(images.shape[0], -1)
        optimizer.zero_grad()
        output = model(images)
        loss = criterion(output, labels)
        loss.backward()
        optimizer.step()
        running_loss += loss.item()
    else:
        train_loss = running_loss / len(trainloader)
        train_losses.append(train_loss)

        test_loss = 0
        accuracy = 0
        with torch.no_grad():
            model.eval()
            for images, labels in testloader:
                images = images.view(images.shape[0], -1)
                output = model(images)
                test_loss += criterion(output, labels).item()
                ps = torch.exp(output)
                top_p, top_class = ps.max(dim=1)
                equals = top_class == labels
                accuracy += torch.mean(equals.type(torch.FloatTensor))

        model.train()
        test_loss /= len(testloader)
        test_losses.append(test_loss)
        test_accuracies.append(accuracy / len(testloader))

        print(f"Epoch: {e+1} - Training loss: {train_loss:.4f}, Test loss: {test_loss:.4f}
```





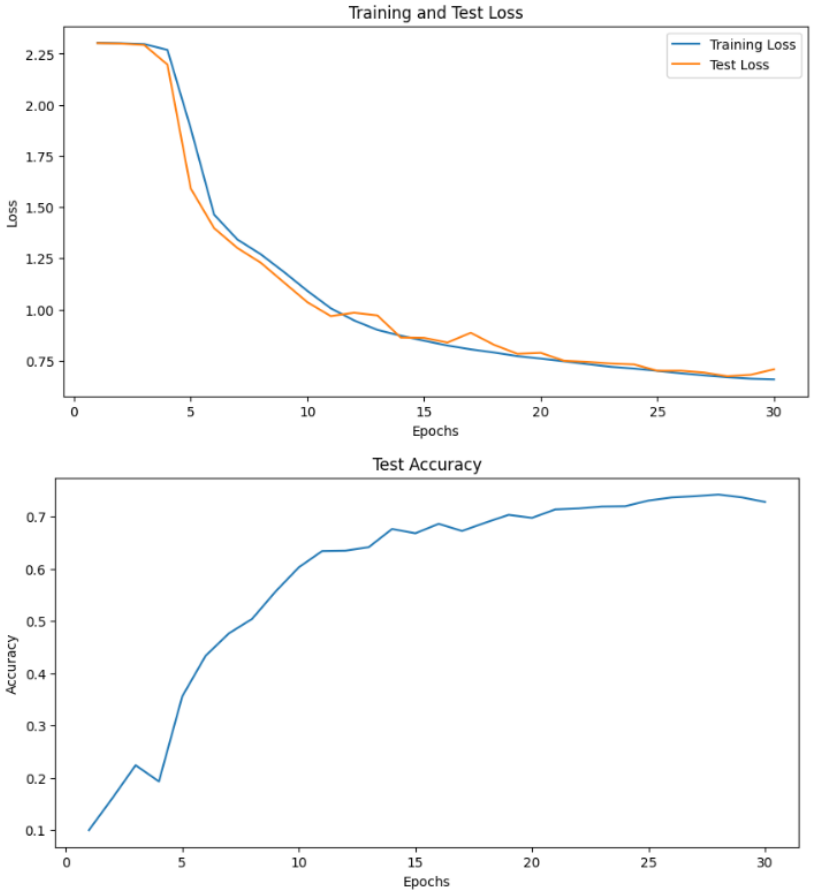
پ)

در این قسمت، مدل و کد های آموزش ونمایش خروجی ها ثابت اند و تنها برای داده افزایی، متغیر مربوط به transform تغییر کرده است.برای اینکار داده افزایی با چرخش عکس برحسب زاویه (10 درجه)کراپ نمودن عکس تغییر در رنگ و ویژگی های رنگی و نوری عکس،چرخش افقی و... انجام دادیم. و سپس آن را هنگام دانلود و پردازش داده آموزش استفاده نمودیم.

```
transform = transforms.Compose([
    transforms.RandomRotation(10),
    transforms.RandomHorizontalFlip(),
    transforms.RandomAffine(degrees=0, translate=(0.1, 0.1)),
    transforms.RandomCrop(size=28, padding=4),
    transforms.ColorJitter(brightness=0.4, contrast=0.4, saturation=0.4, hue=0.1),
    transforms.ToTensor()
])
```

```
# Download and load training data
trainset = datasets.FashionMNIST('./data',download=True, train= True, transform=transform)
trainloader = torch.utils.data.DataLoader(trainset, batch_size= 64, shuffle=True)

# Download and load test data
testset = datasets.FashionMNIST('./data',download=True, train= False, transform=transform)
testloader = torch.utils.data.DataLoader(testset, batch_size= 64, shuffle=True)
```

نتایج خروجی نشان از بهبود عملکرد داده تست می دهد و اینکه در هر دور، عملکرد آن بهبود یافته و تفاوت خطا میان دو بخش کاهش یافته است.همچنین دقت روی داده های تست،به گونه ای است که به صورت صعودی روندی داشته و نوسانی و ثابت شدن در epoch های متوالی در آن رخ نداده است. بدین سان مدل بدرستی آموزش دیده و روی داده های دیده نشده هم عملکرد مناسبی داشته است.

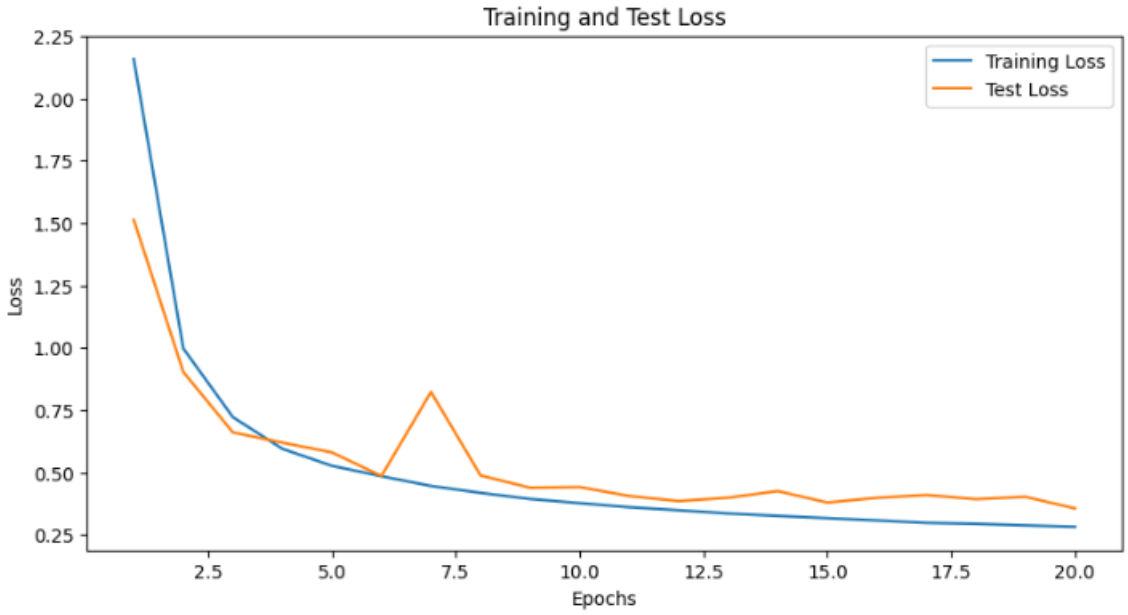


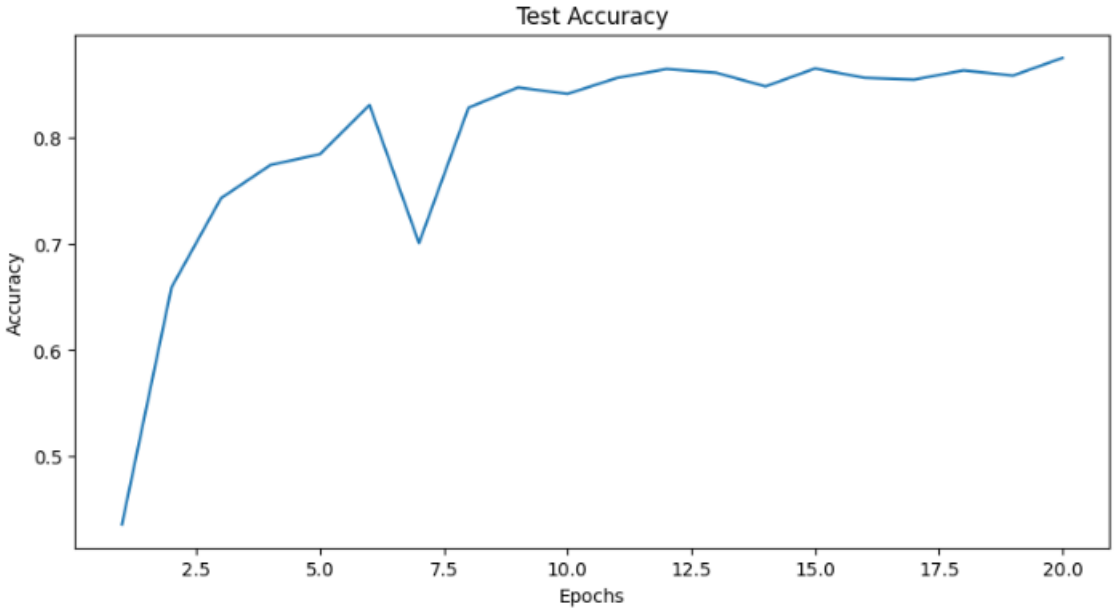
ت)

در این قسمت هم از روش L2 استفاده می نمایم و مقدار weight decay را 0.005 می گذاریم. همین طور که مشخص است مقدار بیش برازش کم شده و مدل خروجی بهتری پیدا کرده است ولی نتیجه آن همانطوز که مشخص است نسبت به داده افزایی بهتر نشده است . البته با تغییر در محدوده متغیر یا استفاده از روش منظم سازی دیگر ممکن است نتیجه ای بهتر از این نیز حاصل شود. پیرامون دقت داده تست هم همانطور که مشخص است در یک محدوده ای دچار

کاهش کیفیت شده است ولی دوباره دقت آن افزایش یافته است.

```
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr=0.01, weight_decay=0.005)
```





پایان :)