بسم الله الرحمن الرحیم

پاسخ مینی پروژه چهارم یادگیری ماشین

Google Colab
GitHub

نگارش: علی شعبانپور مقدم - هدیه شوشیان

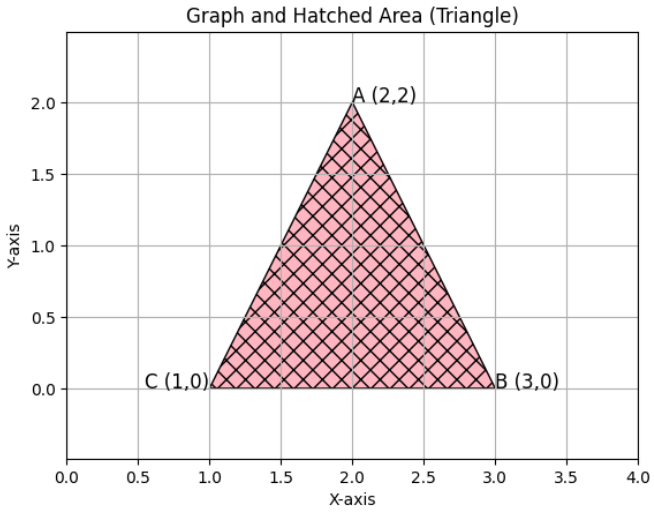
شماره دانشجویی: ۴۰۳۰۸۰۵۴-۴۰۲۰۷۳۰۴

استاد درس: دکتر مهدی علیاری شوره دلی

بهار ۱۴۰۴

فهرست مطالب

۱ پرسش اول



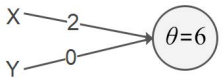
در ابتدا نیاز است شبکه مورد نیاز برای انجام کار طبقه بندی را طراحی کنیم. چون سه شرط در این سوال (شامل سه ضلع مثلث) مطرح است، از ۳ نرون McCulloch-Pitts در لایه اول استفاده می‌کنیم و از یک نرون نهایی برای عمل‌گیری استفاده می‌شود. همان‌طور که گفتیم، ابتدا نیاز است تا وزن‌ها و Treshold سه نرون اول را معلوم کنیم. با بدست آوردن معادلات خط AB، BC، AC به ترتیب داریم:

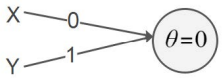
$$2x + y = 6$$

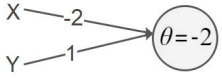
$$y = 0$$

$$-2x + y = -2$$

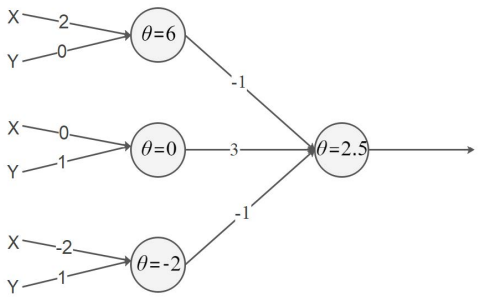
با توجه به روابط و ورودی x و y نمای لایه اول شبکه به صورت زیر خواهد بود.







سپس باید وزن‌های نرون‌های لایه دوم را طوری تنظیم کنیم که وقتی نرون اول خروجی صفر داشت (سمت چپ خط AB) و نرون دوم خروجی ۱ داشت (بالای خط BC) و نرون سوم خروجی صفر (سمت راست خطAC) خروجی ۱ آتش کند. به طوری که باید برای ورودی‌های صفر از لایه قبل جریمه بیشتری اختصاص دهیم. در نهایت شبکه به همراه Treshold به صورت زیر خواهد بود.



حالا میخواهیم شبکه را مدل سازی کنیم ۴ نورون به صورت زیر تعریف میکنیم:

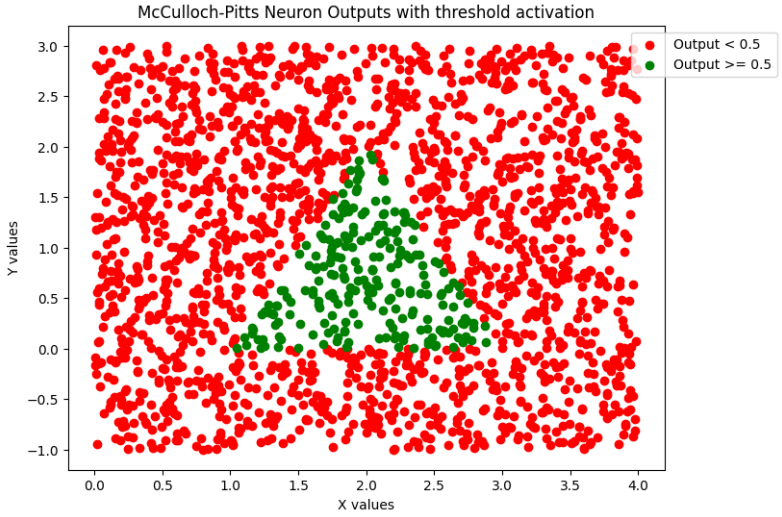
```python
# Define model for dataset
def Area(x, y, activation='threshold'):
    neur1 = McCulloch_Pitts_neuron([2, 1], activation=activation, threshold=6)
    neur2 = McCulloch_Pitts_neuron([0, 1], activation=activation, threshold=0)
    neur3 = McCulloch_Pitts_neuron([-2, 1], activation=activation, threshold=-2)
    neur5 = McCulloch_Pitts_neuron([-1, 3, -1], activation=activation, threshold=2.5)

    z1 = neur1.model(np.array([x, y]))
    z2 = neur2.model(np.array([x, y]))
    z3 = neur3.model(np.array([x, y]))
    z4 = neur5.model(np.array([z1, z2, z3]))

    return z4
```

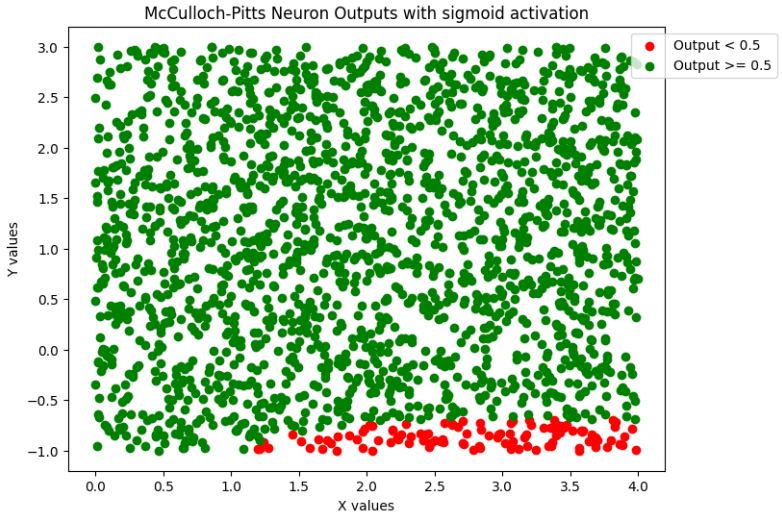
سپس به‌صورت زیر 2000 نقطه در این محدوده تولید می‌کنیم و آن را به عنوان ورودی به شبکه می‌دهیم. باید نقاطی که شبکه به عنوان کلاس 1 در نظر می‌گیرد سبز و بقیه نقاط (به عنوان کلاس صفر) قرمز باشند.

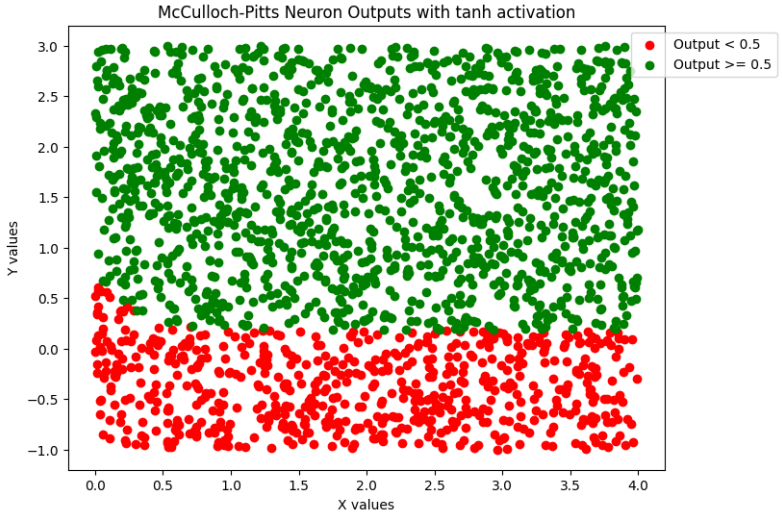
خروجی مدل به صورت زیر است. همان‌طور که دیده می‌شود، به خوبی کار طبقه‌بندی انجام شده و نقاط داخل مثلث (به مرز نواحی دقت شود) همگی با رنگ سبز و لیبل 1 مشخص شده‌اند و نقاط خارج آن با رنگ قرمز و لیبل 0 هستند. در نتیجه، شبکه به خوبی در جدا کردن داده‌ها عمل می‌کند.

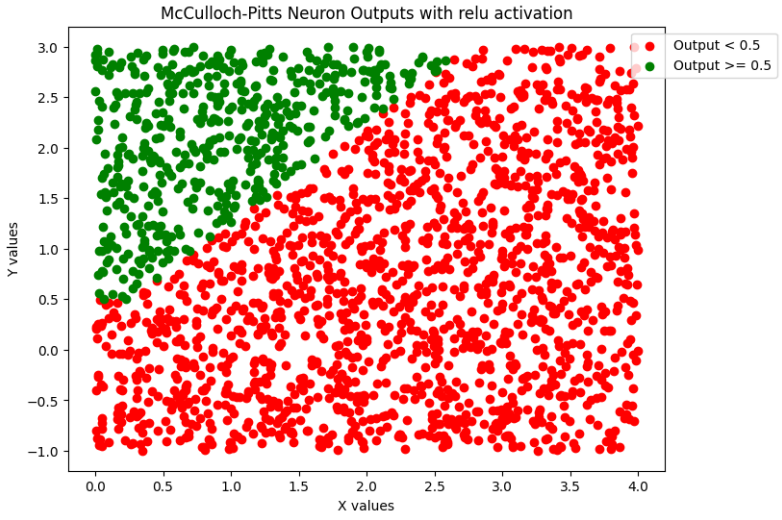


در قسمت بعد، اثر activation functionهای مختلف را در مدل می‌بینیم. برای انجام این کار، یک ورودی به ورودی‌های کلاس McCulloch-Pitts اضافه می‌کنیم. همان‌طور که در بالا دیده می‌شود، یک ورودی به کلاس اضافه شده که در حالتی که به طور خاص تعریف نشود، در حالت activation قرار دارد.

این توابع در قالب سه متد treshhold تعریف شده‌اند و در نهایت با یک دستور classify and plot که با if شرط‌بندی می‌کند، به ما خروجی می‌دهد. در نهایت، یک تابع activation تولید شده که تنها با توجه به ورودی، آن نوع تابع فعال‌ساز را مشخص کرده و با توجه به این تابع، کار رسم را انجام می‌دهد.


McCulloch-Pitts Neuron Outputs with sigmoid activation
Output < 0.5
Output >= 0.5
Y values
X values


McCulloch-Pitts Neuron Outputs with tanh activation
Output < 0.5
Output >= 0.5
Y values
X values


McCulloch-Pitts Neuron Outputs with relu activation
Output < 0.5
Output >= 0.5
Y values
X values

دیده می‌شود که با تغییر توابع فعال‌ساز، عملکرد شبکه به‌طور کلی تحت‌الشعاع قرار می‌گیرد و شبکه باید با درنظرگیری این توابع طراحی شود. در این حالت چون شبکه طراحی‌شده بر مبنای فعال‌ساز treshhold بود، همین فعال‌سازی نیز بهترین گزینه است. در حالی که با فعال‌سازهای دیگر، طبقه‌بند به خوبی کار نمی‌کند و تنها بخشی از کار طبقه‌بندی را انجام می‌دهد.

۲ پرسش دوم

۱.۲ دادگان

۱.۱.۲

در این مقاله چهار مکان از موریس از نظر آب و هوا مورد بررسی قرار گرفته است. پارامترهای آب و هوایی مورد استفاده در این مقاله شامل دما،سرعت باد،جهت باد،فشار، رطوبت و میزان ابری بودن است. داده های مورد استفاده از چندین منبع مختلف جمع آوری شده اند از جمله ایستگاه های هواشناسی محلی و پیش بینی جهانی.داده های ایستگاه هواشناسی محلی از چهار ایستگاه های محلی بطور همزمان جمع آوری شده اند و ویژگی آنها این است که داده هایی دقیق برحسب ساعت یا روز ارائه می دهند. در حالی که داده های جهانی از مدل های پیش بینی آب و هوا از قبیل $ERAS, GFS$ استفاده می کند و اطلاعات آب و هوایی چند روز آینده را با دقت کمتری ارائه می کند.

پس از جمع آوری داده ها برای تشکیل دیتاست ابتدا یکپارچه سازی بر حسب ساعت صورت گرفته است بطوری که اگر فرمت زمان بندی بصورت ساعتی باشد و برای داده های گمشده از میانگین محلی برای جایگزین بهره برده شده است. و دیتاست نهایی ترکیبی از دیتاهای ایستگاه ها و پیش بینی جهانی است.

۲.۱.۲

در دادگان کگل شهرهای

$BASEL, BUDAPEST, DEBILT, DRESDEN, DUSSELDORF, HEATHROW, KASSEL$

$LJUBLJANA, MAASTRICHT, MALMO, MONTELIMAR, MUENCHEN, OSLO, PERPIGNAN$

$SONNBLICK, STOCKHOLM, TOURS, ROMA$ موجود است

که از این بین سه شهر $Montelimar, PerpignanandTours$ تنها از شهرهای فرانسه می باشد.

برای پیش‌پردازش داده‌ها، مقاله ابتدا داده‌ها را به‌صورت ساعتی یکپارچه‌سازی کرده و مقادیر گمشده را با میانگین محلی جایگزین نموده است.

توضیح کد: ابتدا تمامی کتابخانه ها را فراخوانی می کنیم و سپس با دستور $kagglehub.dataset_download$ دیتاست را دانلود کرده و پس از ذخیره در df آن را پرینت می کنیم. سپس ستون هایی از دیتاست را که شامل تاریخ و ماه و ویژگی های آب و هوایی شهرهای فرانسه هستند را در df ذخیره می کنیم.

۳.۱.۲

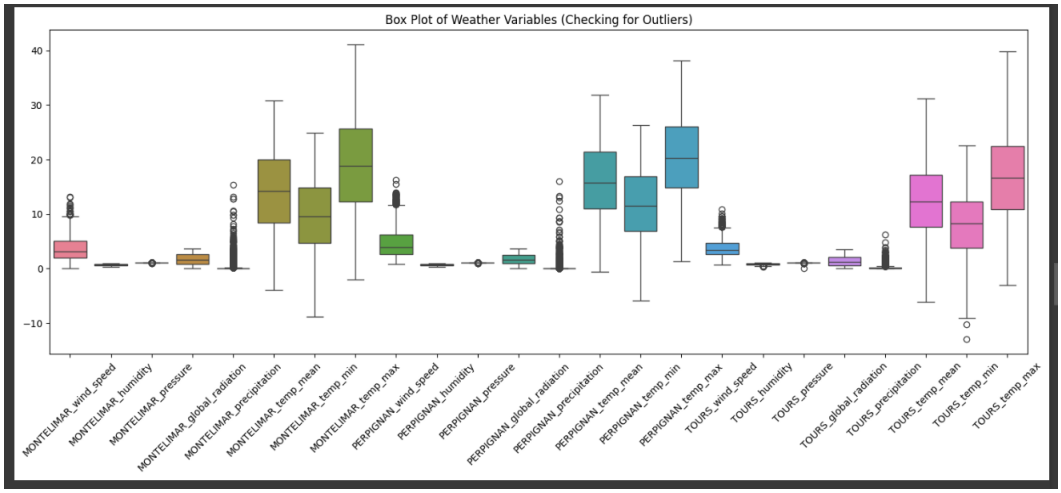
در این دادگان 3654 نمونه به ازای همین تعداد روز از تاریخ $2000/01/01$ تا تاریخ $2010/01/01$ وجود دارد.

توضیح کد: با دستور $pd.to_datetime$ فرمت عددی تاریخ را به فرمت زمانی یا آرایه ای تبدیل می کنیم. و با دستور $df["DATE"].min()$ مقدار مینیمم تاریخ و ماکزیمم آن که نمایانگر زمان شروع و پایان نمونه برداری است را بدست می آوریم.

در این مقاله پیش پردازش های $MissingDataHandling, Min-MaxNormalization, TimeAlignmentorSynchronization, FeatureExtraction, Time-BasedSampling$

انجام شده است. در دادگان ما دیتایی از دست نرفته است پس نیاز به جایگزین داده ی گمشده نمی باشد. اما جنس داده ها از نوع رطوبت و دما و... می باشند پس نیاز به نرمال سازی است. هماهنگ سازی داده ها و نمونه برداری زمانی نیز در دادگان از قبل لحاظ شده است. استخراج ویژگی نیز انجام لحاظ می گردد. پس کافی است نرمال سازی و استخراج ویژگی صورت گیرد. ما در این سوال به منظور درک بهتر استاندارد سازی را نیز انجام دادیم.

پیش از انجام پیش پردازش ها باکس پلات داده های سه شهر فرانسه به ازای هشت ویژگی به شکل زیر بدست آمده است. مشاهده می کنیم به علت تفاوت جنس داده ها و وجود داده های پرت و تراکم داده ها در مقادیر مشخصی می توانیم از و نرمالیزه سازی بهره ببریم.

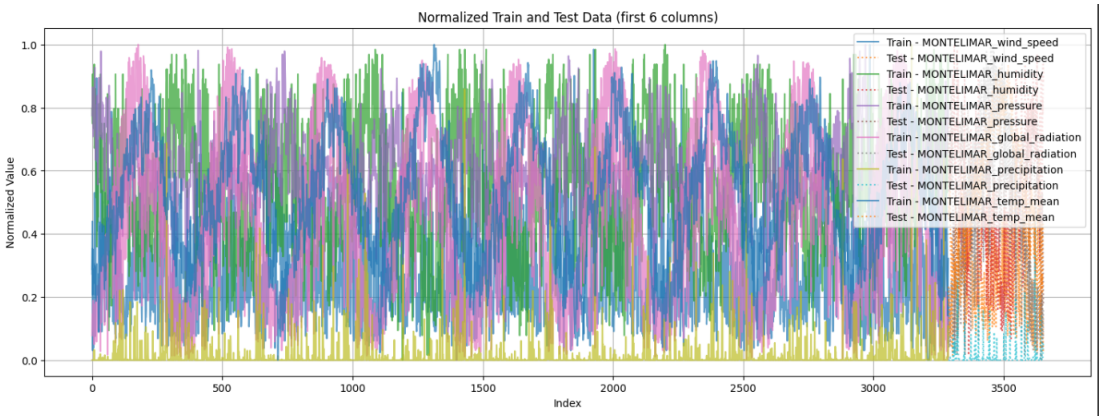


شکل ۱: باکس پلات برای داده بدون پیش پردازش

توضیح کد: با اضافه کردن کتابخانه ی $sklearn.preprocessing$ و دستور $importStandardScaler$ استانداردسازی تعریف میگردد کافی است ستون های عددی df را با دستور $dfstandard.selectdtypes(include = [”number”]).columns$ جدا کرده و ستون تاریخ را که به شکل آرایه تبدیل شده است حذف کنیم. در نهایت با دستور $scaler.fittransform$ دیتاها استاندارد سازی شده و با نام $dfstandard$ دخیره می گردند. حال نرمال سازی داده ها را انجام می دهیم که در مقاله نیز انجام شده است. بطوری که مقادیر دیتا ها اعدادی بین صفر تا یک به خود می گیرند و از فرمول زیر محاسبه می شود:

$$X_{norm} = \frac{(X - X_{min})}{(X_{max} - X_{min})}$$

و خروجی به شکل زیر خواهد بود:



شکل ۲: باکس پلات برای داده پس از نرمال سازی

توضیح کد : با اضافه کردن کتابخانه ی $sklearn.preprocessing$ و با استفاده از دستور $MinMaxScaler$ نرمال سازی تعریف می گردد. حال مانند حالت قبل کافی است این دستور را روی ستون ها عددی اعمال کنیم و ستون ماه که از نوع عددی نیست و ازنوع آرایه هست را جدا در نظر بگیریم. و در نهایت با دستور $scaler.fittransform$ دادگان نرمال سازی شده را در $dfnormal$ دخیره نماییم.

۴.۱.۲

صورت سوال از ما خواسته داده ها را به نمونه هایی پنجره پنج تایی که با یکدیگر همپوشانی چهار دارند تقسیم نماییم. همچنین داده ها فقط برای سال 2009 در نظر گرفته شوند. پس داده ها را از تاریخ 2009/01/01 تا تاریخ 2010/01/01 جدا می نماییم سپس پنجره ی اول داده ی شماره یک تا پنج خواهد بود و برای پنجره ی دوم داده ی دو تا شش درنظر گرفته می شود. و به همین ترتیب پنجره های بعدی را خواهیم داشت.

با توضیح فوق بدیهی است ویژگی های آب و هوایی چهار روز اول را چون هنوز تشکیل یک پنجره نمی دهند نمی توان پیش بینی نمود.

همچنین در بخش دیگری از سوال دمای روزی را برحسب روز قبل نیز خواسته است. پس در کل دو پنجره زمانی خواهیم داشت یکی برای پنج روز دیگری برای یک روز

توضیح کد:پس از آن که داده های آموزش و آزمون را از یکدیگر تفکیک کردیم ستون های غیر عددی در اینجا تاریخ و ماه را نیز حذف می کنیم. سپس یک تابع $generate_sequence_windows$ تعریف می کنیم تا به ازای طول پنجره ی زمانی و تعداد $step$ دیتا فریم مورد نظر را به چندین پنجره تقسیم بندی نماید.در نهایت با دستور $df_france_train.iloc[4:].copy()$ مقادیر نهایی یا هدف را می یابیم.

کد اول: ایجاد توالی‌های لغزشی

```
fed generate_sequence_windows(dataframe, window_length, step=1):
    tropmi numpy as np
    data = dataframe.select_dtypes(include=[np.number]).to_numpy()
    total_rows, num_features = data.shape
    windows = np.lib.stride_tricks.sliding_window_view(data, (window_length, num_features))
    nruter windows.reshape(-1, window_length, num_features)[::step]
```

این تابع داده‌ها را به توالی‌های لغزشی با طول مشخص (window_length) و گام (step) تقسیم می‌کند. خروجی یک آرایه ۳ شامل توالی‌های زمانی است.

کد دوم: آماده‌سازی توالی‌های آموزشی و آزمایشی

```
train_sequences = generate_sequence_windows(df_france_train, window_length=5, step=1)
test_sequences = generate_sequence_windows(df_france_test, window_length=5, step=1)
train_targets = df_france_train.iloc[4:].copy()
test_targets = df_france_test.iloc[4:].copy()
```

توالی‌ها برای داده‌های آموزشی و آزمایشی با طول ۵ ایجاد شده و مقادیر هدف از ردیف‌های بعدی انتخاب می‌شوند تا هم‌راستا با توالی‌ها باشند.

کد سوم: نمایش ابعاد داده‌ها

```
tnirp(train_targets.shape)
tnirp(train_sequences.shape)
tnirp(test_sequences.shape)
tnirp(test_targets.shape)
```

ابعاد داده‌های ورودی و هدف برای اطمینان از تطابق بررسی می‌شود. این ابعاد باید به‌طور صحیح با تعداد توالی‌ها و ویژگی‌ها مطابقت داشته باشد.

خروجی به شکل زیر خواهد بود:

```
# Display shapes for verification
print(train_targets.shape)
print(train_sequences.shape)
print(test_sequences.shape)
print(test_targets.shape)
```

```
(3284, 27)
(3284, 5, 26)
(361, 5, 26)
(361, 27)
```

شکل ۳: سایز داده های آزمون و آموزش

عنوان هدف نهایی دمای فردای شهر تورس است. مشاهده می شود تاریخ $2009/01/01$ جزو داده های آموزش در نظر گرفته نشده است تعداد سطر ها یا نمونه های دادگان آموزش و آزمون به شکل فوق خواهد بود.

۲.۲ آموزش مدل

۱.۲.۲

برای داده های چند منبعی و پراکنده روش یادگیری ماشین گروهی یا همکارانه روشی مناسب است زیرا می تواند از چندین مدل به ازای ایستگاه های مختلف آب و هوایی با همکاری هم و بطور همزمان برای نتیجه ای بهتر بهره ببرد. این مدل ها بطور مستقل از هم اطلاعات خود را با سایر مدل ها به اشتراک می گذارند تا بهترین پیش بین بدست آید. این روش همچنین باعث بهبود امنیت داده ها می شود زیرا داده ها در یک مکان واحد ذخیره نمی گردند.استفاده از منبع توزیع شده هزینه ها را نیر کاهش می دهد زیرا ضرروت نیاز به زیرساخت برای پردازش داده بصورت متمرکز از بین می رود.

در مقاله برای هر یک از ایستگاه های آب و هوایی یک مدل و برای پیش بینی جهانی نیز مدل هایی دیگر داریم که با یکدیگر همکاری می کنند تا ویژگی های آب و هوایی را بهتر پیش بینی کنند. دو نمونه برای مدل های ایستگاه آب و هوایی $LSTM, MLP$ می باشد. همچنین دو نمونه از مدل های پیش بینی جهانی استفاده شده در این مقاله مدل های $ERAS, GFS$ می باشد. به عبارت دیگر نتایج پیش بینی های مختلف با یکدیگر ترکیب شده و پیش بینی نهایی بدست می آید. این ترکیب می تواند بصورت میانگین گیری وزنی ، انتخاب بهترین پیش بینی، استفاده از یک مدل یادگیری ثانویه باشد. این نوع از روش های ماشین لرنینگ برای پیش بینی های بلادرنگ مناسب است زیرا عملکردی سریع تر و دقیق تر دارند.همچنین امنیت داده ها را تضمین می نمایند.

یادگیری ماشینی مشارکتی به روشی گفته می‌شود که در آن چند منبع داده یا نهاد مختلف (مانند دستگاه‌های محلی، سرورها یا کاربران از مناطق گوناگون) به صورت اشتراکی در فرآیند آموزش یا پیش‌بینی یک مدل یادگیری مشارکت می‌کنند. این رویکرد علاوه بر امکان استفاده از داده‌های متنوع، در برخی روش‌ها مانند یادگیری غیرمتمرکز، حریم خصوصی کاربران را نیز حفظ می‌کند؛ زیرا به‌جای تبادل داده‌های خام، تنها اطلاعات به‌روزشده مدل میان منابع مختلف مبادله می‌شود.

از جمله مزایای مهم این روش می‌توان به موارد زیر اشاره کرد:

افزایش دقت پیش‌بینی به دلیل استفاده از داده‌های گسترده و متنوع؛

بهره‌وری بیشتر از منابع محاسباتی از طریق توزیع پردازش بین دستگاه‌ها یا مناطق مختلف؛

قابلیت توسعه‌پذیری بالا در پردازش داده‌های حجیم؛

حفاظت از اطلاعات کاربران به‌ویژه در ساختارهای غیرمتمرکز.

در مطالعه انجام‌شده در این مقاله، از یادگیری مشارکتی برای پیش‌بینی کوتاه‌مدت وضعیت هوا در جزیره موریس استفاده شده است. داده‌های آب‌وهوایی نظیر دما، رطوبت، سرعت و جهت باد، فشار و میزان پوشش ابر از چهار منطقه موکا، کورپایپ، واکاس و کواتر بورن، در بازه زمانی اول تا بیست‌ویکم اردیبهشت ۱۴۰۰، با نرخ چهار نمونه در هر ساعت، گردآوری شده‌اند. این داده‌ها از طریق دستگاه‌های محلی مانند تلفن همراه یا رایانه جمع‌آوری و به سامانه‌های محلی یا سامانه‌های ابری منتقل شده‌اند.

در بخش پیاده‌سازی، ساختار سامانه به‌صورت مشارکتی طراحی شده است؛ یعنی دستگاه‌های محلی، یک سامانه مرکزی در محل، و فضای ابری به صورت هماهنگ داده‌ها را پردازش کرده و مدل را آموزش می‌دهند. برخلاف روش‌های سنتی که کاملاً متمرکز هستند، این ساختار نیمه‌متمرکز امکان پردازش داده‌ها به‌صورت توزیع‌شده را فراهم کرده است. همچنین داده‌ها در پایگاه‌های داده محلی یا ابری ذخیره شده و برای آموزش مدل‌ها مورد استفاده قرار گرفته‌اند.

در مرحله آموزش، اطلاعات مناطق مختلف به‌صورت ترکیبی برای آموزش مدل‌های پیش‌بینی به‌کار رفته‌اند. این تلفیق داده‌ای به شناسایی بهتر الگوهای آب‌وهوایی منجر شده و دقت پیش‌بینی را برای هر منطقه افزایش داده است. طبق نتایج ارائه‌شده، مدل مشارکتی نسبت به مدل‌های آموزش‌دیده با داده‌های تک‌منطقه‌ای عملکرد دقیق‌تری داشته و خطای پیش‌بینی میانگین آن بسیار پایین (نزدیک به صفر) گزارش شده است.

برای آموزش و ارزیابی مدل‌ها از روش پنجره لغزنده استفاده شده است؛ در این روش، داده‌ها به صورت گام‌به‌گام به مدل داده می‌شوند و از آن برای پیش‌بینی مقادیر آینده استفاده می‌گردد. داده‌ها نیز در قالب آرایه‌های چندبعدی ساماندهی شده‌اند تا برای ورود به مدل‌های یادگیری مورد استفاده قرار گیرند.

در مجموع، این رویکرد مشارکتی موجب شده تا مدل به اطلاعات گسترده‌تری دسترسی داشته باشد و از نظر دقت و سرعت پیش‌بینی عملکرد بهتری از خود نشان دهد.

۳.۲

برای حل این مسئله، شهر **مونتلیمار** به‌عنوان نقطه هدف انتخاب شده و یک مدل یادگیری ماشین با ساختار ساده شامل یک لایه پنهان طراحی می‌شود. ورودی این مدل شامل دنباله‌های زمانی از داده‌های نرمال‌شده مربوط به چند شهر فرانسه (مونتلیمار، پرپینیان و تور) است که با پنجره‌ای به اندازه ۵ و با هم‌پوشانی ۴ استخراج شده‌اند. خروجی مدل، پیش‌بینی چند ویژگی جوی مربوط به شهر مونتلیمار مانند سرعت باد، رطوبت و ... خواهد بود.

برای یادگیری، سه مدل با نرخ‌های گام متفاوت (یکی زیاد، یکی متوسط و یکی بسیار کوچک) طراحی شده‌اند. فرایند آموزش با استفاده از روش کاهش تدریجی خطا انجام می‌شود، تابع سنجش خطا از نوع میانگین توان دوم اختلاف، و تابع‌های فعال‌سازی نیز به‌ترتیب ریلو در لایه میانی و خطی در لایه خروجی هستند. آموزش برای ۲۰۰ دوره انجام شده و در هر مرحله، میزان خطای مدل روی داده‌های آموزش و آزمون اندازه‌گیری و ثبت می‌شود. هم‌چنین، نمودار تغییرات خطا و نوار پیشرفت فرآیند آموزش نیز برای تحلیل بهتر عملکرد مدل ترسیم می‌گردد.

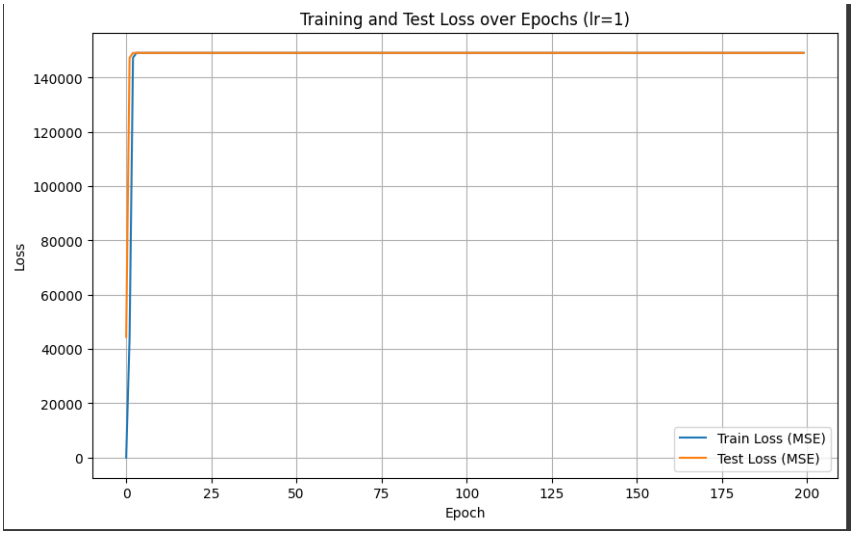
```
Using device: cpu

Training with learning rate = 1
LR=1 Training Progress: 100%|██████████| 200/200 [00:04<00:00, 43.83it/s]
Epoch [1/200] - Train Loss: 1.222329 - Test Loss: 44438.253906
Epoch [2/200] - Train Loss: 44338.980469 - Test Loss: 147322.593750
Epoch [3/200] - Train Loss: 147333.843750 - Test Loss: 149136.250000
Epoch [4/200] - Train Loss: 149137.921875 - Test Loss: 149139.640625
Epoch [5/200] - Train Loss: 149137.953125 - Test Loss: 149136.281250
Epoch [6/200] - Train Loss: 149137.984375 - Test Loss: 149139.671875
Epoch [7/200] - Train Loss: 149138.000000 - Test Loss: 149136.359375
Epoch [8/200] - Train Loss: 149138.031250 - Test Loss: 149139.718750
Epoch [9/200] - Train Loss: 149138.046875 - Test Loss: 149136.390625
Epoch [10/200] - Train Loss: 149138.078125 - Test Loss: 149139.718750
Epoch [11/200] - Train Loss: 149138.046875 - Test Loss: 149136.390625
Epoch [12/200] - Train Loss: 149138.078125 - Test Loss: 149139.718750
Epoch [13/200] - Train Loss: 149138.046875 - Test Loss: 149136.390625
Epoch [14/200] - Train Loss: 149138.078125 - Test Loss: 149139.718750
```

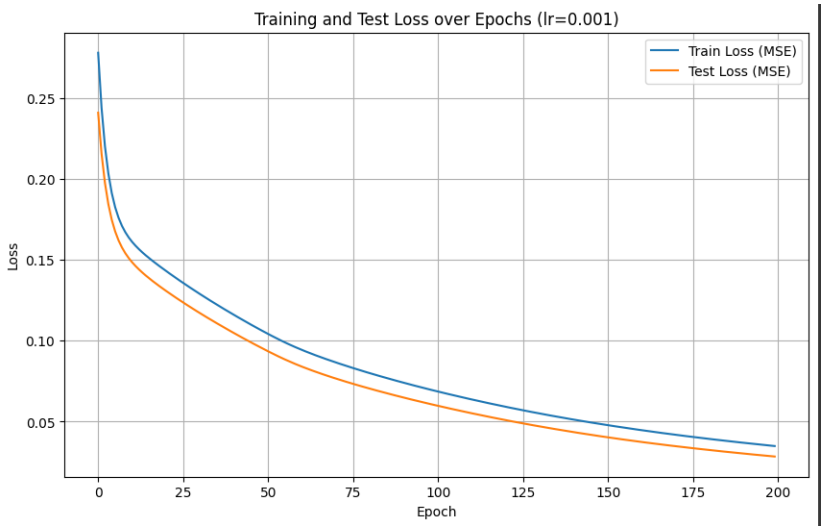
شکل ۴: خروجی

۴.۲

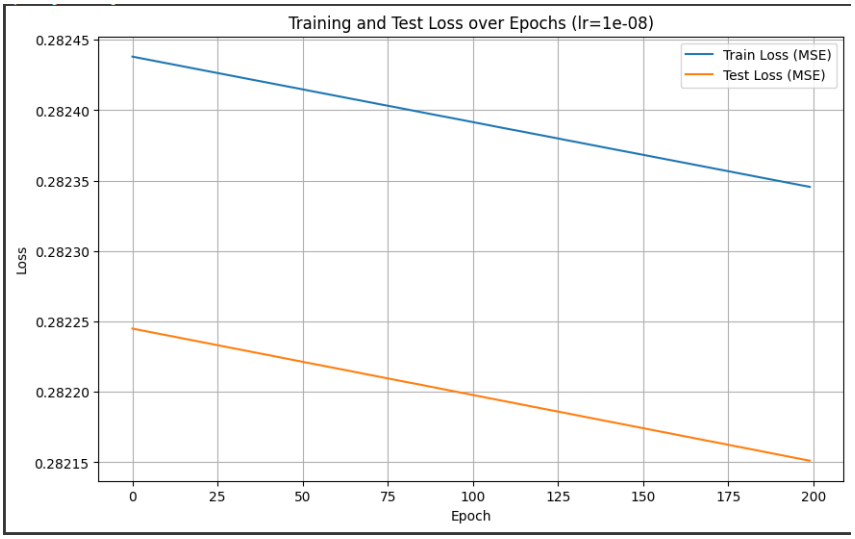
۱.۴.۲



شکل ۵

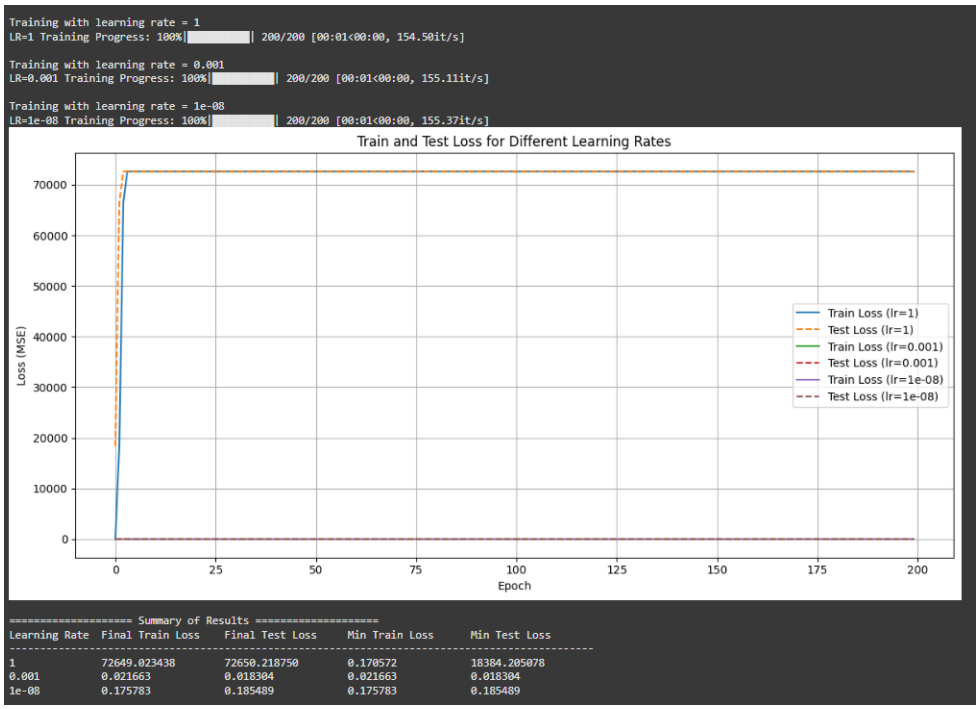


شکل ۶



شکل ۷

۲.۴.۲



شکل ۸

۳.۴.۲ توضیح جامع دو بخش فوق

تحلیل جامع نمودارهای خطای آموزش و آزمون برای نرخ‌های مختلف یادگیری نمودار فوق، روند تغییر مقدار خطا (اتلاف) در مرحله‌ی آموزش و آزمون را برای یک شبکه‌ی عصبی با نرخ‌های یادگیری مختلف نمایش می‌دهد. در ادامه، تحلیل دقیقی از دلایل نامناسب بودن نرخ‌های بسیار بالا و بسیار پایین، و اهمیت انتخاب یک مقدار مناسب برای نرخ یادگیری ارائه می‌گردد:

نرخ یادگیری بالا (مثلاً برابر با ۱): وقتی نرخ یادگیری بیش از حد بالا باشد، مقدار خطا به مقدار «نامشخص» (یا به اصطلاح عددی غیرقابل‌محاسبه) تبدیل می‌شود. این پدیده نشان‌دهنده‌ی ناپایداری شدید در فرآیند یادگیری است. علت آن معمولاً این است که مدل هنگام به‌روزرسانی وزن‌ها، گام‌هایی بسیار بزرگ برمی‌دارد و از مسیر بهینه‌سازی منحرف می‌شود. در نتیجه، نه‌تنها به سمت کمینه‌ی تابع هزینه حرکت نمی‌کند، بلکه از آن دور می‌شود و فرآیند یادگیری ناکام می‌ماند.

به بیان ساده‌تر، مقدار بسیار زیاد برای نرخ یادگیری باعث می‌شود مدل پیوسته از حالت تعادل خارج شده و نتواند الگوهای داده را به‌درستی یاد بگیرد. این موضوع در نمودار با افزایش ناگهانی مقدار خطا یا نوسانات شدید دیده می‌شود که به‌وضوح نشانه‌ی شکست در یادگیری است.

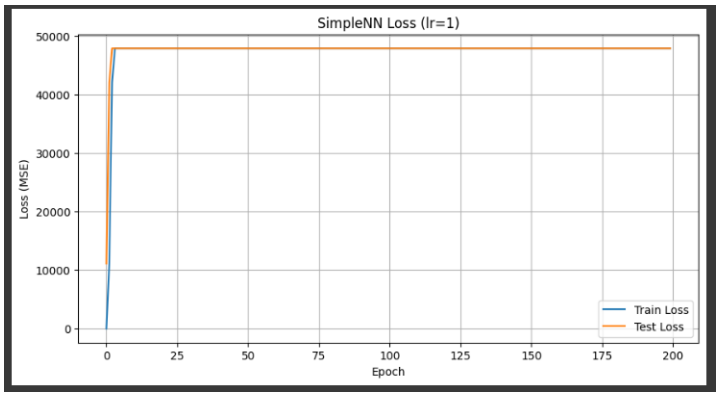
نرخ یادگیری بسیار پایین (مثلاً برابر با ۰٫۰۰۰۰۰۰۰۱): در این حالت، مقادیر خطای آموزش و آزمون در طول تکرارها تقریباً ثابت باقی مانده و کاهش چندانی ندارند. دلیل این رفتار آن است که مقدار تغییرات وزن‌ها در هر مرحله‌ی یادگیری بسیار ناچیز است و عملاً مدل در جای خود باقی می‌ماند. در نتیجه، فرآیند یادگیری یا بسیار کند می‌شود یا عملاً متوقف می‌گردد. در چنین شرایطی، مدل توانایی یادگیری الگوهای معنادار از داده‌ها را ندارد و عملکرد آن بهبود نمی‌یابد. این وضعیت از نظر زمانی و محاسباتی به‌شدت ناکارآمد است و نمی‌تواند به نتیجه‌ی مطلوب منجر شود.

نرخ یادگیری مناسب (مثلاً برابر با ۰٫۰۰۱ یا ۰٫۰۱): نمودار نشان می‌دهد که در نرخ‌های یادگیری میانه، مدل به‌صورت یکنواخت و پیوسته در حال کاهش مقدار خطا در هر دو مرحله‌ی آموزش و آزمون است. این وضعیت نشان‌دهنده‌ی یادگیری مؤثر، به‌روزرسانی‌های متعادل و نزدیک‌شدن تدریجی به حالت بهینه است. در چنین حالتی، گام‌های یادگیری نه آن‌قدر بزرگ هستند که باعث ناپایداری شوند و نه آن‌قدر کوچک که از پیشرفت جلوگیری کنند. این تعادل باعث می‌شود مدل هم در آموزش خوب عمل کند و هم بتواند به‌خوبی برای داده‌های جدید تعمیم یابد.

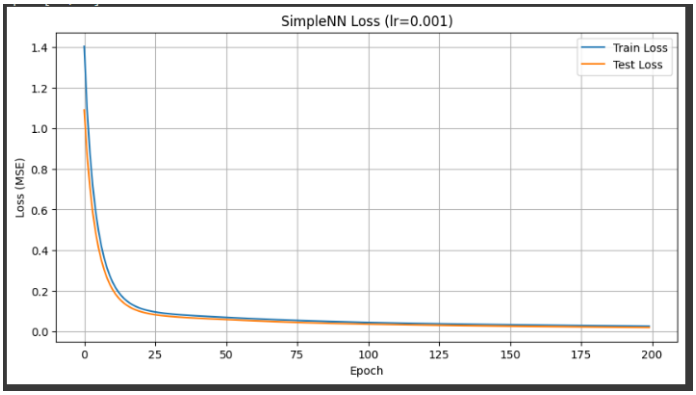
جمع‌بندی کلی

نرخ‌های یادگیری بسیار بالا موجب ناپایداری و عدم همگرایی می‌شوند و نرخ‌های بسیار پایین باعث می‌شوند فرآیند یادگیری تقریباً متوقف شود. انتخاب یک نرخ یادگیری مناسب (مانند ۰٫۰۰۱ یا ۰٫۰۱) برای رسیدن به آموزش پایدار، کاهش مؤثر خطا و عملکرد مناسب مدل ضروری است. این انتخاب نقش کلیدی در موفقیت الگوریتم یادگیری دارد و می‌تواند به بهبود دقت و کارایی مدل کمک شایانی نماید.

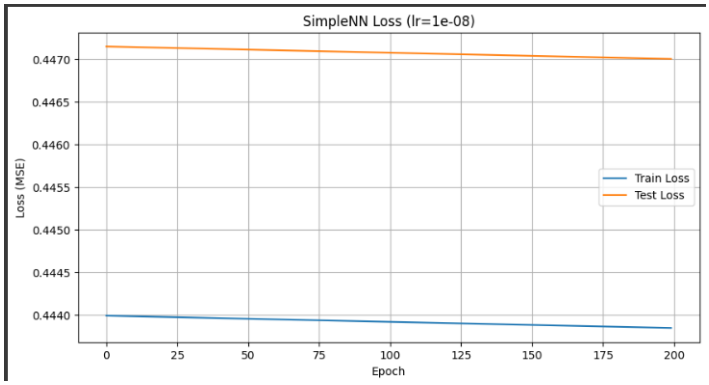
۵.۲



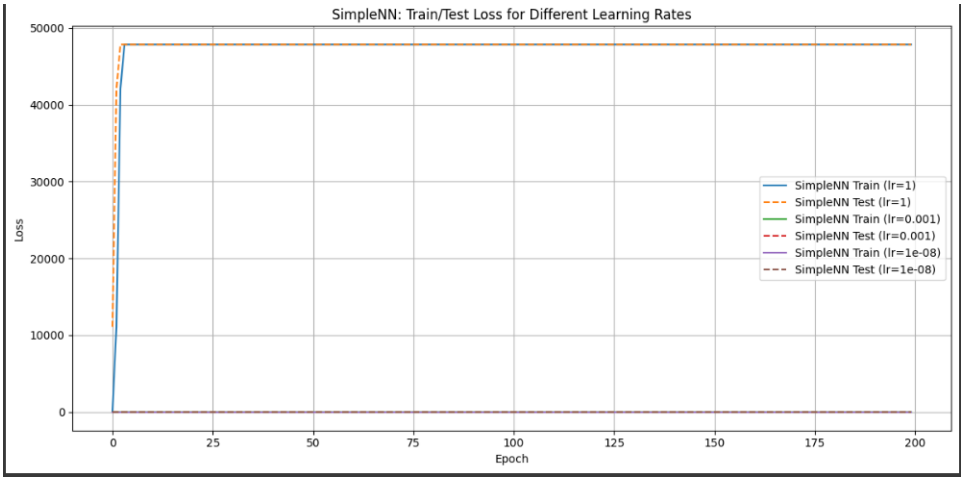
شکل ۹



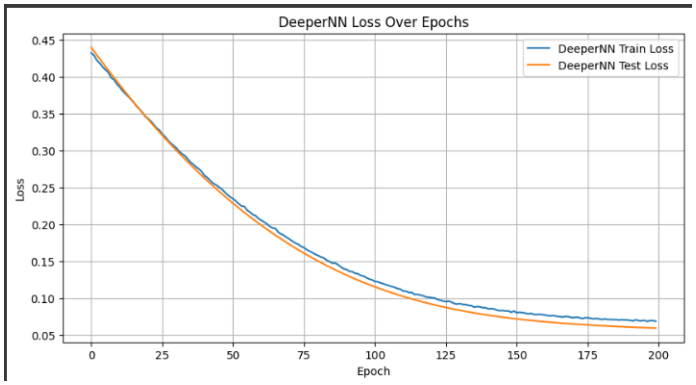
شکل ۱۰



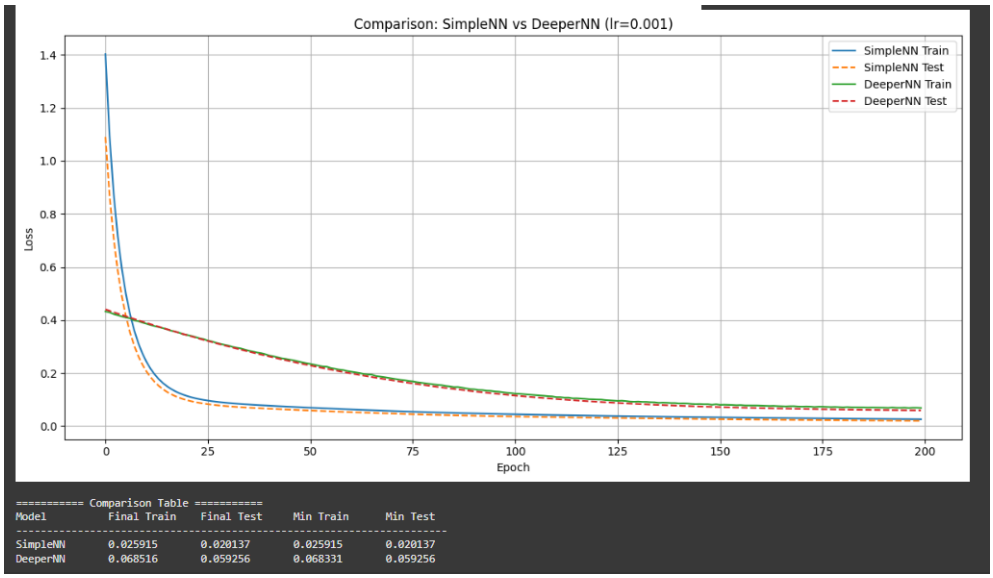
شکل ۱۱



شکل ۱۲



شکل ۱۳



شکل ۱۴

تمامی نمودار های خروجی شبکه عصبی ساده تر و شبکه عصبی عمیق تر و مقایسه ی بین ان دو به شکل های فوق است.

در این مرحله، با هدف بررسی تأثیر افزایش تعداد لایه‌های پنهان در شبکه عصبی، دو لایه‌ی پنهان به ساختار مدل افزوده شده است تا بتوان عملکرد مدل را در حالت ساده‌تر و عمیق‌تر مقایسه کرد. از آن‌جا که در مرحله‌ی قبل مشاهده شد نرخ یادگیری ۰٫۰۰۱ بهترین نتیجه را در کاهش خطای آموزش و آزمون به همراه داشته است، این مقدار به عنوان نرخ ثابت برای آموزش مدل در این مرحله در نظر گرفته شده است. تغییر در معماری شبکه موجب شد روند یادگیری در هر دو حالت مورد بررسی قرار گیرد و بتوان درباره‌ی اثرات مثبت و منفی افزایش عمق شبکه تحلیل دقیق‌تری ارائه داد.

در حالت ساده‌تر، نمودار خطا از مقدار نسبتاً بالایی آغاز می‌شود و با پیشرفت دوره‌های آموزشی، کاهش محسوسی را نشان می‌دهد. این رفتار بیانگر آن است که اگرچه مدل در ابتدا قدرت بیان کافی برای درک داده‌ها را نداشته، اما به مرور توانسته الگوها را فرا گرفته و خطا را کاهش دهد. در مقابل، در حالت عمیق‌تر، مدل از مقدار خطای اولیه‌ی پایین‌تری شروع می‌کند که نشانه‌ای از توان بیشتر آن در شناسایی ویژگی‌های داده‌ها در ابتدای فرآیند یادگیری است. با این حال، با ادامه‌ی آموزش، کاهش خطا کندتر شده و در پایان، مقدار خطای آن از حالت ساده‌تر بیشتر باقی مانده است. این مشاهده می‌تواند به چند دلیل باشد: احتمال گیر افتادن در نقاط بهینه‌ی ضعیف‌تر، همگرایی ناقص به دلیل پیچیدگی مدل، یا حتی بروز بیش‌پردازش به دلیل تعداد زیاد پارامترها.

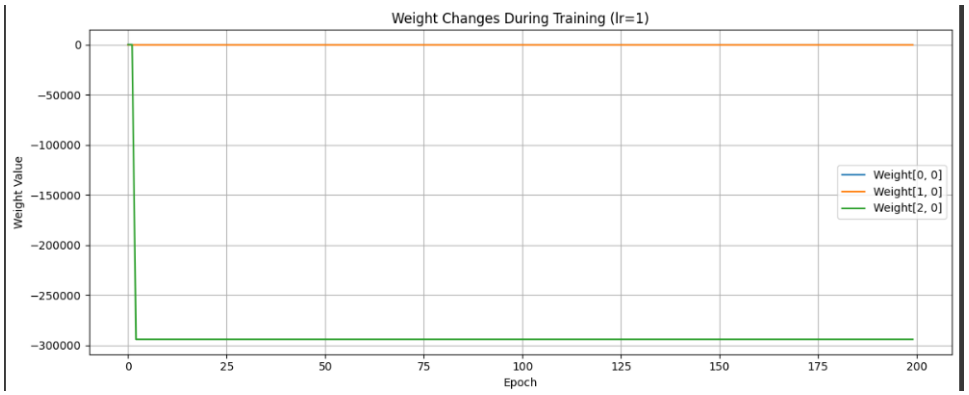
افزایش عمق شبکه به‌طور کلی امکان یادگیری الگوهای پیچیده‌تر و انتزاعی‌تر را فراهم می‌کند و در بسیاری از موارد می‌تواند به بهبود عملکرد منجر شود. در این آزمایش نیز دیده می‌شود که خطای آموزش و آزمون در هر دو مدل کاهش می‌یابد و در مدل عمیق‌تر حتی به حدود ۰٫۰۳ در پایان آموزش می‌رسد، که در

نگاه اول نشان‌دهنده‌ی بهبود یادگیری است. اما نکته‌ی مهم در تفسیر این نتیجه آن است که در نمودار مدل عمیق‌تر، تفاوت میان خطای آموزش و آزمون در برخی نقاط قابل توجه است. این اختلاف می‌تواند نشان‌دهنده‌ی آغاز پدیده‌ی بیش‌پردازش باشد؛ حالتی که در آن مدل بیش از حد به داده‌های آموزش وابسته شده و توان تعمیم خود را به داده‌های جدید از دست می‌دهد.

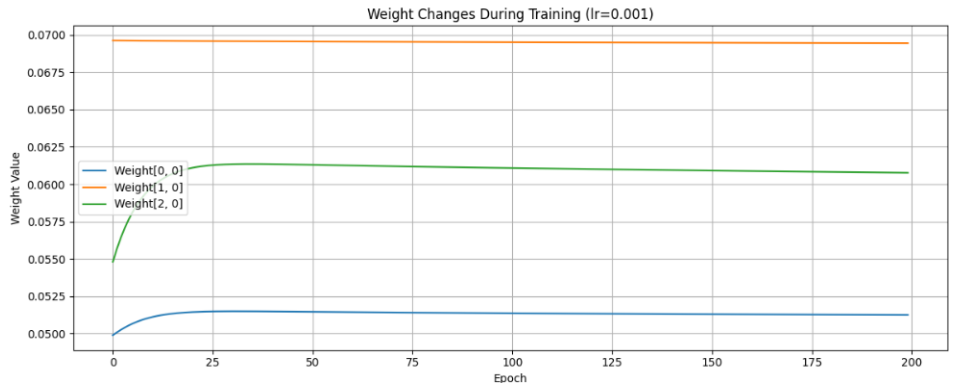
در چنین شرایطی، انتخاب عمق مناسب برای شبکه اهمیت بسیار زیادی دارد. اگر عمق خیلی کم باشد، مدل قادر به یادگیری روابط پیچیده‌ی میان داده‌ها نخواهد بود و در اصطلاح دچار کم‌پردازش می‌شود. از سوی دیگر، اگر عمق بیش از حد زیاد شود، مدل گرایش به حفظ ویژگی‌های خاص داده‌های آموزش پیدا می‌کند و در مواجهه با داده‌های جدید عملکرد ضعیف‌تری نشان می‌دهد. برای دستیابی به یک عملکرد متعادل و پایدار، لازم است که طراحی شبکه با دقت انجام شود و از تکنیک‌هایی مانند تنظیم‌پذیری، داده‌های اعتبارسنجی، و توقف زودهنگام استفاده گردد تا از بروز بیش‌پردازش جلوگیری شود.

نکته‌ی جالب در این بررسی آن است که اگرچه مدل عمیق‌تر با مزیت اولیه‌ی خطای کمتر شروع می‌کند، اما به دلایلی مانند نیاز به زمان بیشتر برای همگرایی، افزایش احتمال بروز نوسان در گرادیان‌ها یا نبود روش‌های تنظیم مؤثر، در پایان آموزش عملکرد بهتری نسبت به مدل ساده‌تر از خود نشان نمی‌دهد. این نتیجه بیانگر آن است که عمیق‌تر بودن شبکه، به‌تنهایی و بدون تنظیمات دقیق و کنترل‌شده، الزاماً به معنای بهبود عملکرد نهایی نیست.

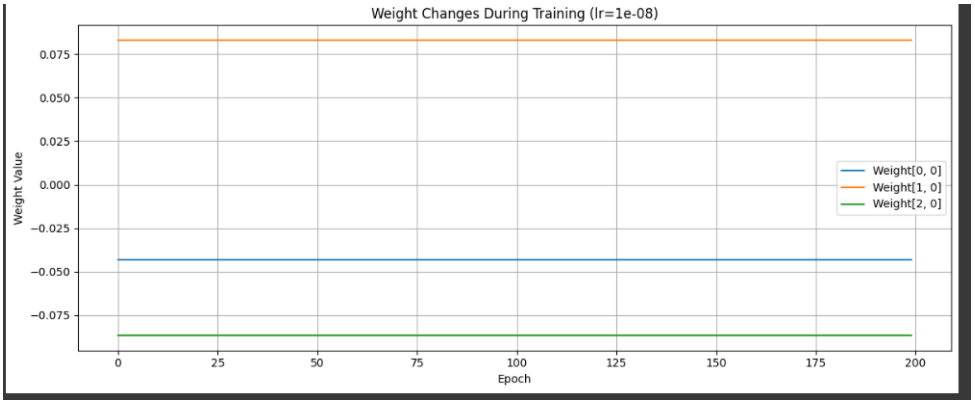
۶.۲



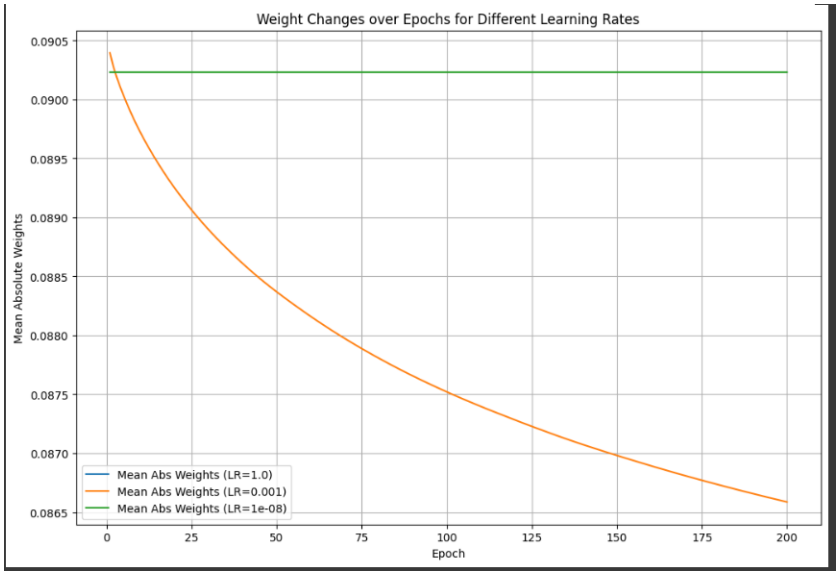
شکل ۱۵



شکل ۱۶



شکل ۱۷



شکل ۱۸

کل وزن‌های لایه پنهان رو در هر ایپاک ذخیره می‌کنیم، به‌صورت یک آرایه دوبعدی. بنابراین تغییرات هر وزن به‌تنهایی رو مشاهده می کنیم برای سه عکس ابتدایی نحوه محاسبه بصورتی است که : تغییرات وزن‌های لایه‌ی مخفی شبکه‌ی عصبی را در طول فرایند آموزش برای چند نرخ یادگیری مختلف بررسی و ذخیره می‌کند. در هر دوره آموزش، وزن‌های لایه‌ی مخفی جداگانه استخراج و در دیکشنری‌ای ذخیره می‌شوند تا روند تغییر آن‌ها طی اپک‌ها قابل پیگیری باشد. سپس برای هر نرخ یادگیری، تغییرات سه وزن اول مربوط به ورودی اول آن لایه به صورت نمودار نمایش داده می‌شود تا بتوان روند تغییر وزن‌ها را مشاهده کرد. همچنین میانگین قدر مطلق تمام وزن‌های لایه‌ی مخفی در هر دوره محاسبه و ذخیره می‌شود تا بتوان به صورت عددی میزان تغییر و رفتار کلی وزن‌ها را با توجه به نرخ یادگیری مقایسه کرد. به این ترتیب تحلیل وزن‌ها محدود به وزن‌های لایه‌ی مخفی بوده و تمرکز اصلی بر روی نحوه تغییر این وزن‌ها در طول آموزش با نرخ‌های یادگیری متفاوت است.

برای بررسی تأثیر نرخ یادگیری بر تغییرات وزن‌ها در طول آموزش، میانگین مقدار مطلق وزن‌ها در هر دوره‌ی آموزشی محاسبه و ذخیره شده است. از آن‌جا که تعداد وزن‌های شبکه زیاد است، استفاده از میانگین باعث شده تا روند تغییرات به شکل ساده‌تری در نمودار نمایش یابد. نمودار به‌دست‌آمده، تغییرات میانگین مطلق وزن‌ها را در طول دوره‌های آموزش برای نرخ‌های یادگیری مختلف نشان می‌دهد و به‌خوبی تأثیر نرخ یادگیری بر روند به‌روزرسانی وزن‌ها را آشکار می‌سازد.

در حالت نرخ یادگیری بسیار بالا، مانند مقدار یک، تغییرات وزن‌ها به شکلی قابل‌محاسبه یا معنادار در نمودار دیده نمی‌شود. این رفتار نشان‌دهنده‌ی ناپایداری

در به‌روزرسانی وزن‌ها است. زمانی که نرخ یادگیری بیش از حد بزرگ باشد، گام‌های اصلاح وزن به‌قدری بزرگ می‌شوند که وزن‌ها ممکن است دچار نوسانات شدید یا مقادیر غیرقابل‌قبول شوند. در چنین شرایطی مدل از مسیر یادگیری منحرف شده و فرآیند آموزش عملاً از کنترل خارج می‌شود. این موضوع باعث می‌شود مدل نتواند به نقطه‌ی بهینه نزدیک شود و همگرایی رخ ندهد.

در سوی دیگر، زمانی که نرخ یادگیری بسیار پایین انتخاب می‌شود، مانند مقدار بسیار کوچک در حد ده به توان منفی هشت، تغییرات وزن‌ها در طول آموزش تقریباً ثابت و نزدیک به صفر باقی می‌مانند. این رفتار نشان می‌دهد که اصلاحات اعمال‌شده بر وزن‌ها به‌قدری کم هستند که تأثیر محسوسی در روند یادگیری ندارند. در نتیجه، مدل توانایی تطبیق با داده‌ها را از دست می‌دهد و یادگیری عملاً متوقف می‌شود.

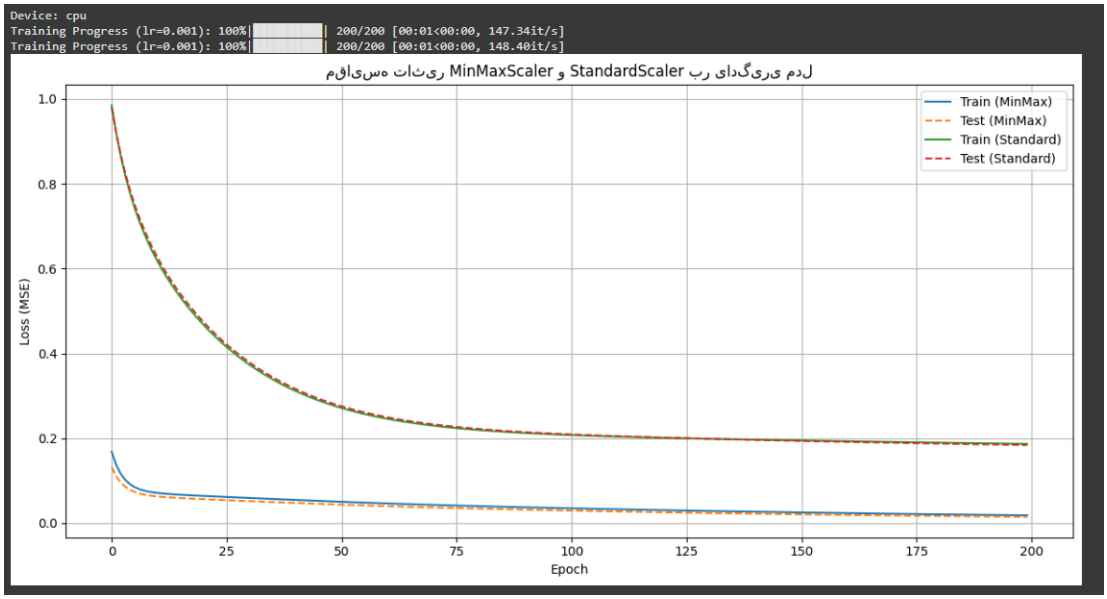
در مقابل، وقتی نرخ یادگیری در حد متعادلی مانند هزارم انتخاب می‌شود، تغییرات وزن‌ها در ابتدا محسوس و قابل‌توجه است و سپس به‌تدریج کاهش می‌یابد و در پایان به مقدار پایداری می‌رسد. این روند نشان می‌دهد که وزن‌ها به‌شکل مؤثر و منظم به‌روزرسانی می‌شوند و مدل به‌سوی وضعیت بهینه حرکت می‌کند. در این حالت، نه نوسانات شدید وجود دارد و نه توقف در یادگیری مشاهده می‌شود. چنین رفتاری نشانه‌ای از انتخاب درست نرخ یادگیری و فراهم شدن شرایط مناسب برای یادگیری پایدار و دقیق است.

در مجموع، می‌توان نتیجه گرفت که نرخ یادگیری یکی از مهم‌ترین عوامل مؤثر در تنظیم تغییرات وزن‌ها در طول آموزش است. مقدار بیش از حد زیاد آن موجب ناپایداری و مقدار بیش از حد کم آن موجب توقف یادگیری می‌شود. در حالی‌که یک مقدار متعادل، فرآیند یادگیری را به‌صورت گام‌به‌گام، قابل‌کنترل و موفق پیش می‌برد. بنابراین انتخاب درست نرخ یادگیری نقشی کلیدی در بهینه‌سازی مدل و دستیابی به عملکرد قابل‌قبول دارد.

در آموزش شبکه‌های عصبی، تنظیم مناسب دیگر پارامترهای مؤثر نیز اهمیت بالایی دارد. از جمله عواملی که در ادامه مورد بررسی قرار خواهند گرفت، می‌توان به نحوه‌ی پیش‌پردازش داده‌ها، مقداردهی اولیه‌ی وزن‌ها، روش‌های نرمال‌سازی، استفاده از تکنیک‌های تنظیم‌پذیری و به‌کارگیری روش‌های جلوگیری از بیش‌پردازش اشاره کرد که هر یک به نوبه‌ی خود در بهبود عملکرد شبکه و دستیابی به نتایج دقیق‌تر نقش دارند. توضیحات تکمیلی درباره‌ی این عوامل در ادامه ارائه خواهد شد.

پیش‌پردازش داده‌ها

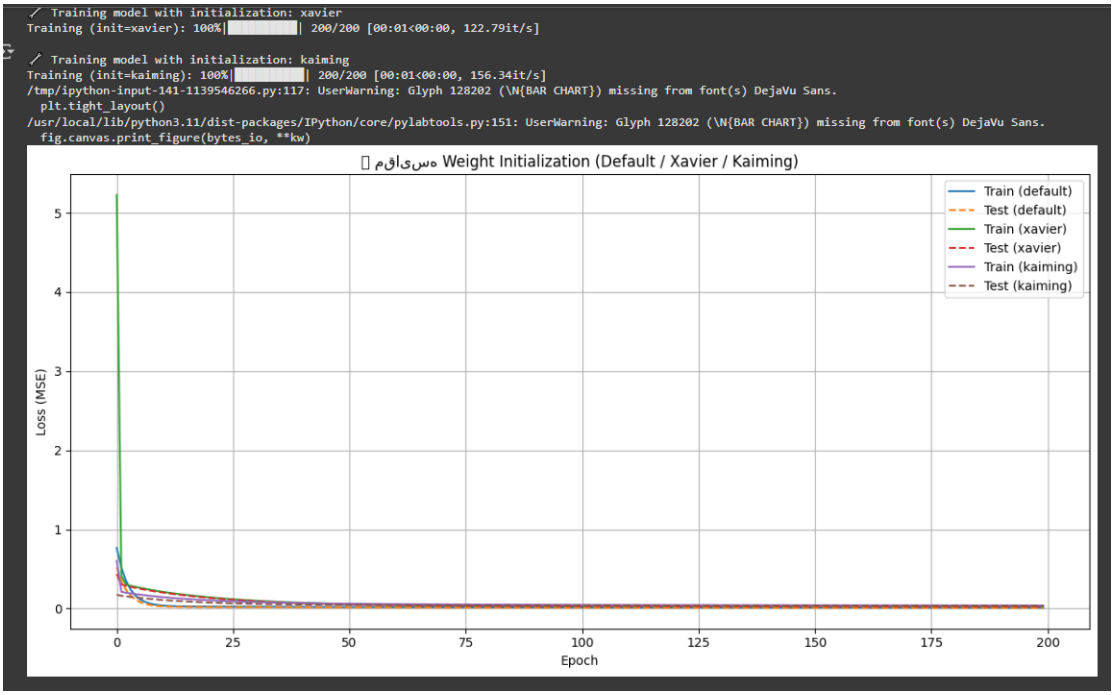
پیش‌پردازش داده‌ها یکی از مراحل کلیدی در یادگیری ماشینی است که شامل حذف نویز، هم‌مقیاس‌سازی ویژگی‌ها، و تنظیم توزیع داده‌ها می‌شود. انجام این مرحله به شکل مناسب باعث می‌شود مدل سریع‌تر و دقیق‌تر آموزش ببیند. برای مثال، زمانی که داده‌ها به صورت عددی در یک بازه مشخص (مثل ۰ تا ۱) نرمال‌سازی شوند، محاسبه گرادیان‌ها در الگوریتم‌های بهینه‌سازی ساده‌تر شده و مدل زودتر به نتیجه پایدار می‌رسد. همچنین، استفاده از روش‌هایی مثل استانداردسازی بر پایه میانگین و انحراف معیار باعث می‌شود داده‌ها توزیع یکنواخت‌تری داشته باشند که در یادگیری عمیق، به‌ویژه برای شبکه‌های دارای چندین لایه، اثر مثبتی بر کاهش خطا و افزایش کارایی دارد.



شکل ۱۹: اثر پیش پردازش

مقداردهی اولیه وزن‌ها

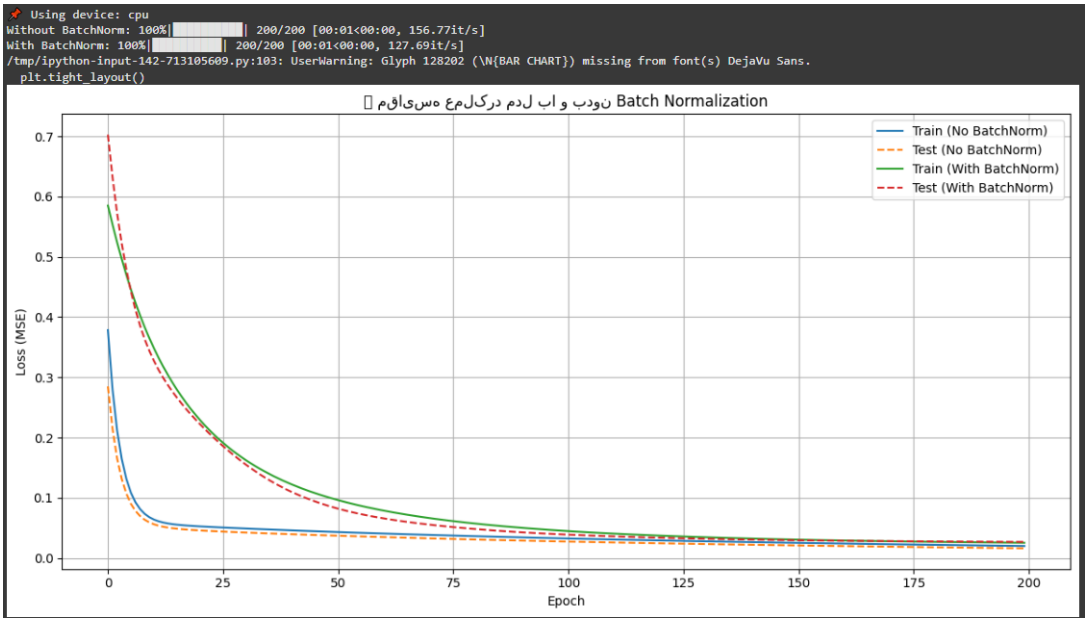
یکی از عوامل مهم در عملکرد دقیق و سریع مدل‌های یادگیری، شیوه‌ی مقداردهی اولیه به وزن‌هاست. اگر این مقداردهی به‌درستی انجام نشود، ممکن است مقدار گرادیان‌ها در هنگام آموزش بسیار بزرگ یا بسیار کوچک شود و در نتیجه روند یادگیری دچار مشکل شود. برای جلوگیری از این مسئله، روش‌هایی وجود دارد که مقدار اولیه‌ی وزن‌ها را با توجه به نوع تابع فعال‌سازی و تعداد نورون‌ها تنظیم می‌کنند. به‌عنوان نمونه، در مدل‌هایی که از تابع فعال‌سازی نوع «رِلو» استفاده می‌کنند، استفاده از مقداردهی مناسب می‌تواند باعث شود که مدل با سرعت بیشتری به نتیجه برسد و تا حدود ۲۰ تا ۳۰ درصد زمان آموزش کاهش یابد. این شیوه باعث می‌شود سیگنال‌ها در میان لایه‌های شبکه بدون کاهش یا افزایش شدید منتقل شوند و یادگیری به شکلی پایدار پیش برود.



شکل ۲۰: اثر مقدار دهی اولیه وزن ها

نرمال‌سازی دسته‌ای

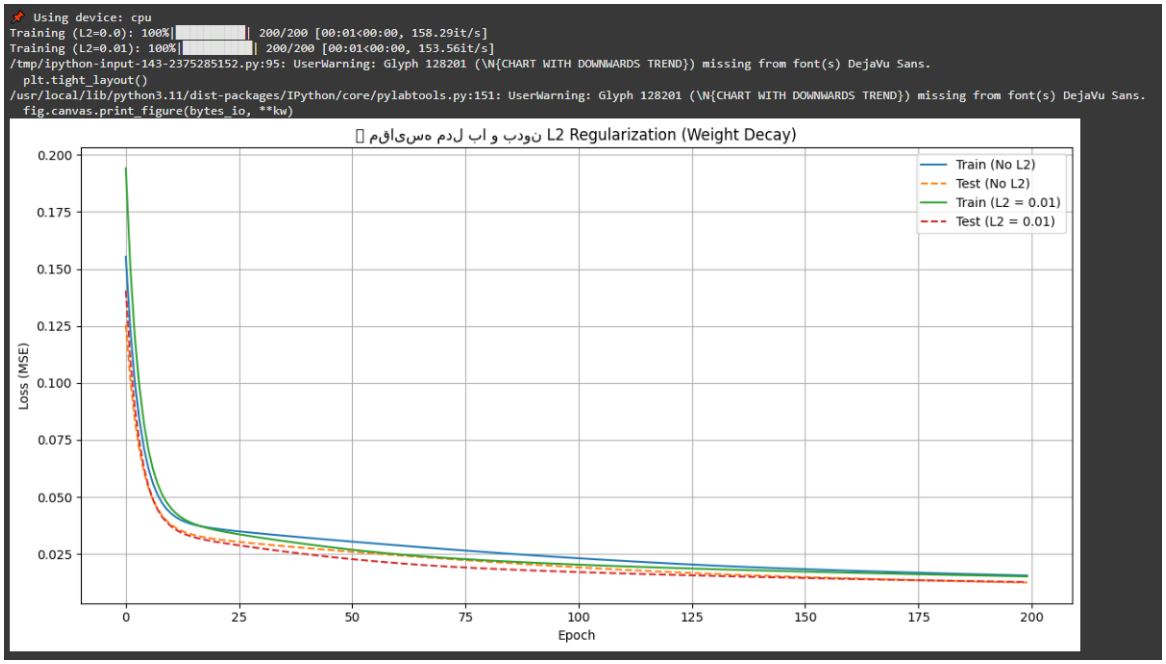
در این روش، میانگین و پراکندگی هر دسته از داده‌ها به گونه‌ای تنظیم می‌شود که توزیع آن‌ها یکنواخت‌تر گردد. این کار باعث می‌شود روند آموزش مدل پایدارتر باشد و امکان استفاده از نرخ‌های یادگیری بزرگ‌تر فراهم شود. در نتیجه، مدت زمان لازم برای رسیدن به نتیجه کاهش می‌یابد و توانایی مدل در تعمیم به داده‌های جدید نیز بهتر می‌شود. بهترین کارایی معمولاً زمانی حاصل می‌شود که اندازه‌ی هر دسته از داده‌ها بین ۱۶ تا ۶۴ انتخاب شود و پارامترهای تنظیم این نرمال‌سازی نیز به‌درستی تعیین شده باشند. در این حالت می‌توان انتظار داشت که مقدار خطا تا حدود ۱۰ تا ۱۵ درصد کاهش یابد.



شکل ۲۱: اثر نرمال سازی دسته ای

تنظیم پذیری

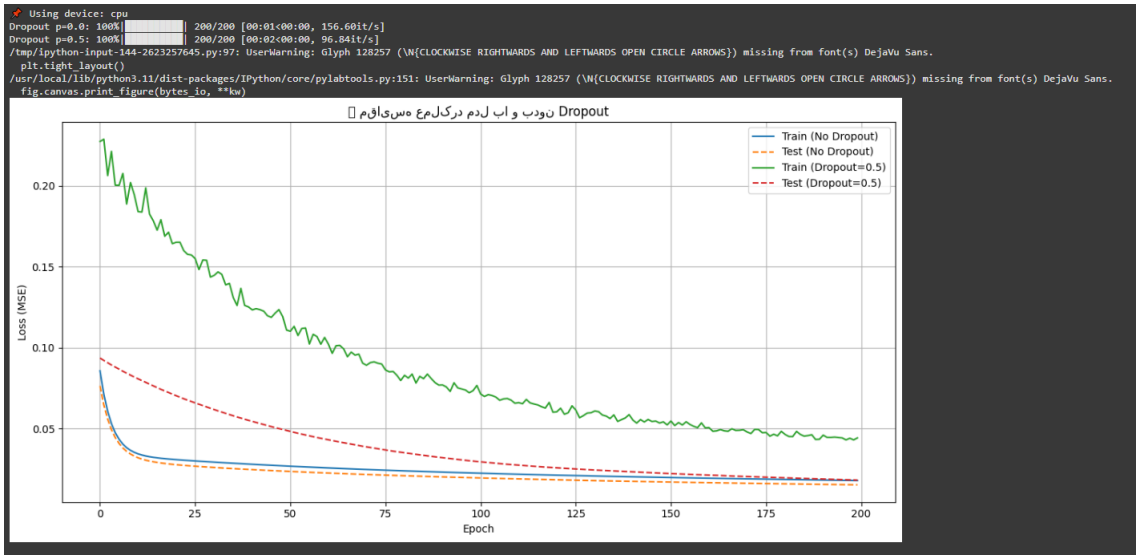
روش‌های کنترل پیچیدگی مدل، مانند اضافه کردن جریمه برای وزن‌های بزرگ، باعث می‌شوند مدل از یادگیری بیش‌ازحد جزئیات داده‌ها جلوگیری کند. این کار باعث می‌شود که مدل بتواند بهتر برای داده‌های جدید عمل کند و دچار انحراف نشود. زمانی بهترین نتیجه حاصل می‌شود که مقدار این جریمه در بازه‌ای میان ۰۰۰۱.۰ تا ۰۱.۰ تنظیم گردد؛ چنین تنظیمی می‌تواند تعادلی میان سادگی مدل و دقت آن ایجاد کرده و خطر بیش‌پردازش را به شکل چشم‌گیری کاهش دهد.



شکل ۲۲: اثر تنظیم پذیری

غیرفعال‌سازی تصادفی نورون‌ها

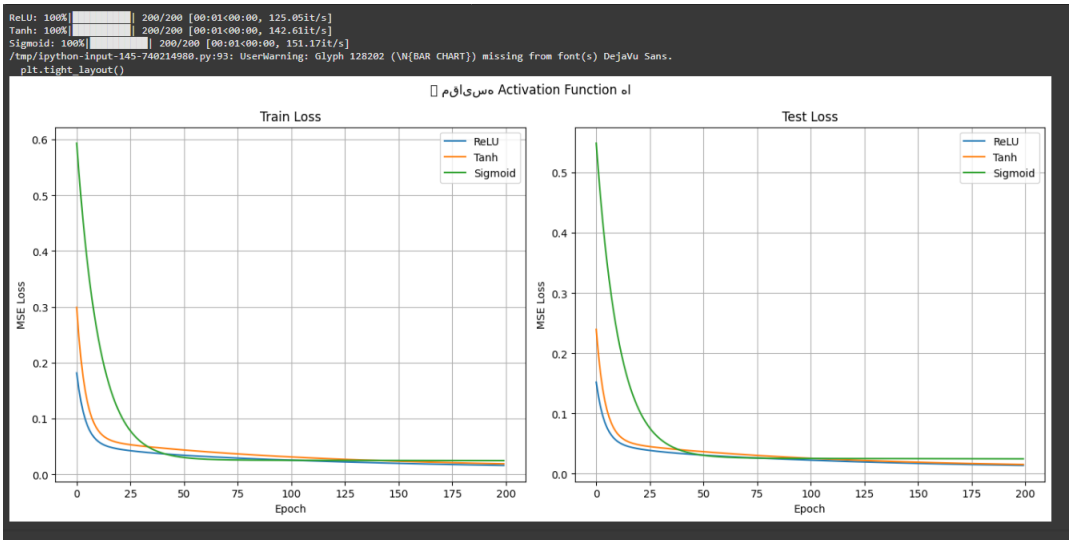
یکی از روش‌های جلوگیری از وابستگی شدید مدل به برخی مسیرهای خاص، غیرفعال‌سازی تصادفی تعدادی از واحدهای پردازشی در هر مرحله یادگیری است. این کار کمک می‌کند تا مدل بتواند ویژگی‌های گوناگون داده را از مسیرهای مختلف یاد بگیرد و به یک ساختار مقاوم‌تر برسد. زمانی بیشترین بهره‌وری حاصل می‌شود که میزان این غیرفعال‌سازی در بازه‌ای میان ۲.۰ تا ۵.۰ تنظیم شود؛ در این حالت، توانایی مدل برای تعمیم روی داده‌های جدید افزایش یافته و دقت آن روی نمونه‌های آزمایشی می‌تواند تا ۵ تا ۱۰ درصد بهتر شود.



شکل ۲۳: اثر غیرفعالسازی تصادفی نورون ها

غیرفعال‌سازی تصادفی نورون‌ها

یکی از روش‌های جلوگیری از وابستگی شدید مدل به برخی مسیرهای خاص، غیرفعال‌سازی تصادفی تعدادی از واحدهای پردازشی در هر مرحله یادگیری است. این کار کمک می‌کند تا مدل بتواند ویژگی‌های گوناگون داده را از مسیرهای مختلف یاد بگیرد و به یک ساختار مقاوم‌تر برسد. زمانی بیشترین بهره‌وری حاصل می‌شود که میزان این غیرفعال‌سازی در بازه‌ای میان ۲.۰ تا ۵.۰ تنظیم شود؛ در این حالت، توانایی مدل برای تعمیم روی داده‌های جدید افزایش یافته و دقت آن روی نمونه‌های آزمایشی می‌تواند تا ۵ تا ۱۰ درصد بهتر شود.



شکل ۲۴

تابع‌های فعال‌سازی مثل ریلو، سیگموید و تانژانت هیپربولیک، نقش کلیدی در شبکه‌های عصبی دارن و کمک می‌کنن مدل بتونه روابط غیرخطی بین ورودی و خروجی رو یاد بگیره. تابع ریلو مقدارهای منفی رو صفر می‌کنه و مقدارهای مثبت رو همون‌طور نگه می‌داره؛ ساده و سریع‌الاجراست و باعث میشه یادگیری سریع‌تر باشه، ولی ممکنه بعضی نورون‌ها خاموش بشن (یعنی همیشه صفر بمونن). تابع سیگموید مقدارها رو به بازه بین صفر تا یک می‌بره و بیشتر در لایه‌های خروجی استفاده می‌شه، اما مشکلش اینه که گرادیان‌ها در مقدارهای خیلی بزرگ یا خیلی کوچیک نزدیک صفر می‌شن و یادگیری رو کند می‌کنن.

تابع تانژانت هیپربولیک شبیه سیگمویده ولی خروجی‌هاش بین منفی یک تا مثبت یکه و معمولا نسبت به سیگموید عملکرد بهتری داره چون خروجی‌هاش نسبت به مرکز صفر متقارن هستن. این تابع‌ها مستقیما روی شکل نمودار اتلاف و دقت آموزش و آزمون تأثیر می‌ذارن و می‌تونن باعث همگرایی بهتر یا گیر کردن مدل در نقطه‌های بد بشن.

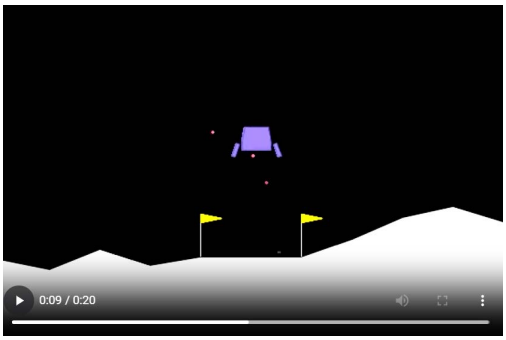
جمع بندی

تنظیم درست این پارامترها وابسته به ساختار شبکه، ویژگی‌های داده‌ها و هدف نهایی آموزش است. انتخاب مقدارهای مناسب می‌تونه باعث بشه مدل سریع‌تر به جواب برسه، دقت بالاتری داشته باشه و از بیش‌برازش جلوگیری کنه. برای رسیدن به بهترین نتیجه، باید به‌صورت پیوسته آزمایش و ارزیابی انجام بشه تا ترکیب بهینه پارامترها پیدا بشه.

۴ پرسش چهارم

قسمت اول

محیط Lunar Lander یک محیط توسعه یافته توسط Open AI در قالب یک کتابخانه متن باز به نام Gym است. از این قالب برای شبیه سازی ها و اهداف آموزشی برای آموزش Reinforcement Learning استفاده می شود. این کتابخانه شامل محیط های زیادی است که هریک مارا با جنبه ای از RL آشنا می کند.مسائل برگرفته از کنترل کلاسیک مانند Cart Pole تا بازی آتاری و حتی شبیه سازی های پیچیده تر مانند MuJoCo. از خوبی های این منبع متن باز؛ یکپارچه بودن آن در تمامی فضا هاست به طوری که تمامی فضاها و نتایج مربوط به آن با هم قابل مقایسه هستند. متد هایی مانند close()، reset()و یا step(action) از جمله این متد ها هستند. حالا که با فضای کلی، توسعه دهنده و اهداف این کتابخانه متن باز آشنا شدیم به سراغ یکی از محیط های آن یعنی Lunar Lander می رویم. شکل ۱ تصویری از این محیط نشان می دهد که در ان یک فضاپیما قصد دارد روی یک مساحت مشخص فرود بیاید. در واقع این یک مسئله بهینه سازی مسیر این راکت یا فضاپیما است که یک مساله کلاسیک است. با توجه به اصل Pontryagin's maximum حالتی بهینه است که موتور در حالت حداکثری خود روشن باشد یا خاموش شود. به همین علت هر موتور دو حالت دارد؛ روشن باشد یا خاموش باشد.



شکل ۱: محیط Lunar Lander

Action Space در این بازی شامل چهار اکشن گسسته است:

- کاری انجام نشود
- موتور جهت چپ روشن شود
- موتور اصلی روشن شود
- موتور جهت راست روشن شود

State Space در این بازی شامل یک بردار ۸ بعدی است.

- پوزیشن X که می تواند بین -1.5 تا 1.5 باشد.
- پوزیشن Y که می تواند بین -1.5 تا 1.5 باشد.

- سرعت خطی در جهت X که می تواند بین -5 تا 5 باشد.
- سرعت خطی در جهت Y که می تواند بین -5 تا 5 باشد.
- در جه که می تواند بین -3.14 تا 3.14 (به رادیان) باشد.
- سرعت زاویه ای که می تواند بین -5 تا 5 باشد.
- یک متغیر بولین که نشان می دهد پای ۱ با زمین ارتباط دارد یا نه
- یک متغیر بولین که نشان می دهد پای ۲ با زمین ارتباط دارد یا نه

Reward System به این صورت است که بازیگر یا فضاپیما را ترغیب می کند تا به صورت دقیق و نرم در محل موردنظر فرود بیاید. پاداش برای حرکت از بالا و فرود آمدن روی سکوی مورد نظر ۱۰۰ تا ۱۴۰ امتیاز است. اگر فضاپیما از سکویی که باید روی آن فرود بیاید فاصله بگیرد پاداش از دست می دهد. اگر تصادف بکند -۱۰۰ امتیاز از دست می دهد. اگر روی سکو بنشیند و بی حرکت بماند یا rest انجام دهد ۱۰۰ امتیاز دریافت می کند. اگر هر پا با زمین برخورد کند ۱۰ امتیاز می گیرد. راه افتادن موتور اصلی -0.03 امتیاز در هر فریم را در بر دارد و برای موتور های کناری نیز همین مقدار است.اینکار برای تشویق به استفاده کمتر از سوخت انجام می گیرد. رسیدن به حالت solved ۲۰۰ امتیاز دارد.
پوزیشن ابتدایی فضاپیما در مرکز و بالای تصویر است و در ابتدا یک نیروی رندوم به مرکز جرم آن اعمال می شود.
هر اپیزود بنابر یکی از دلایل زیر تمام می شود:

- فضاپیما تصادف کند. یعنی بدن آن با ماه برخورد کند.
- فضاپیما خارج از فضای دید بشود برای مثال حرکت افقی آن بیشتر از ۱ شود.
- اگر فضاپیما بیدار نباشد، یعنی حرکتی نداشته باشد و با چیز دیگری برخورد نکند.

قسمت دوم

برای حل این قمست از کد قرار گرفته به عنوان نمونه استفاده شده و تغییراتی در آن اعمال شده تا ارور های آن برطرف شود. در مرحله اول ابتدا باید موارد و کتابخانه های لازم را نصب کنیم. دو نکته وجود دارد که قابل اشاره هستند، اول اینکه باید دو دستور برای نگرفتن ارور به کد اضافه کنیم که در برنامهٔ ۱ آمده اند.

```
!apt-get update
!apt-get install -y swig
```

Code 1: installing swig

دوم برای نشان دادن ویدیو ها به صورت مجازی باید از Xvbf استفاده کنیم که نیاز است تا آن را در محیط google collab نصب کنیم که به وسیله برنامهٔ ۲ اینکار انجام شده است.
سایر دستورات در بخش نصب ثابت باقی مانده است و به سراغ مرحله بعد می رویم.
در مرحله بعد دستور مورد نیاز برای استفاده از GPU در محیط google collab را به صورت برنامهٔ ۳ وارد می کنیم که در نتیجه به ما می گوید که ما در حال استفاده از Coda و در حال ران کردن GPU هستیم.

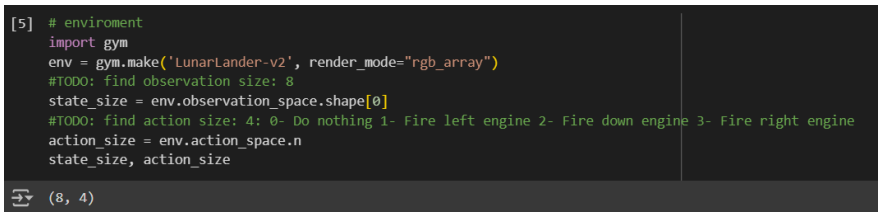
```
!sudo apt-get update
!sudo apt-get install xvfb
```

Code 2: installing Xvbf

```
import torch
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
device
```

Code 3: using GPU

در ادامه به صورت شکل ۲ محیط مورد نظر که همان Lunar Lander است را از Gym وارد می کنیم. همانطور که مشاهده می شود ما ۸ observation یا state داریم و می توانیم ۴ action انجام دهیم که همگی در بخش ۱.۲ توضیح داده شد. در مرحله بعد ابتدا یک صفحه نمایش مجازی تعریف می کنیم و سایز آن را مشخص می کنیم و سپس یک



شکل ۲: محیط import Lunar Lander from Gym

تابع تعریف می کنیم تا بتوانیم ویدیو هارا که در آینده قصد ضبط آن هارا داریم در محیط Jupiter notebook ببینیم. انجام اینکار در برنامهٔ ۴ آمده است.

در مرحله بعد برنامهٔ ۵ را داریم. در ابتدای آن یک تاپل می سازیم که حالت یک experience را در خود ذخیره کند. این موارد شامل حالتی که فضاپیما در آن وجود دارد (قبل از انجام اکشن)، اکشنی که انجام می دهد، حالت بعدی که با انجام اکشن به آن می رود، پاداشی که دریافت می کند و یک متغیر بولین که نشان می دهد آیا عمل تمام شده یا نه است. سپس کلاس ExperienceReplay یک reply buffer را پیاده می کند. در واقع تمامی experience های گذشته یا همان transitions را در خود ذخیره می کند تا در مرحله آموزش از آنها استفاده شود. برای اینکار از یک deque یا double-ended queue استفاده می شود تا memory ساخته شود. reply buffer به agent اجازه می دهد تا از تجربیات گذشته خود با سمپل کردن به صورت رندوم از batch های transition یاد بگیرد و این رندوم سازی باید پایداری یادگیری می شود و موضوع correlation بین experience های متوالی را برای ما بهبود می دهد. store trans ترنزیشن های جدید را به reply buffer اضافه می کند.

در مرحله بعد شبکه عصبی عمیق مربوط به الگوریتم DQN را تشکیل می دهیم. برنامهٔ ۶ این شبکه را نشان می دهد که در کلاس DeepQNetwork تعریف شده است. همانطور که مشخص است این یک شبکه sequential است که نسبت به کد اصلی به صورت مستقیم پیاده سازی شده و برگردانده می شود. ورودی این شبکه به اندازه state ها نرون دارد و در نهایت خروجی این شبکه همان Q value ها هستند که مقادیر انتظاری از پاداش آینده را برای هر اکشن به ما می دهد. پس خروجی این شبکه به اندازه اکشن ها است و مقادیر Q برای آن ها تخمین زده می شود. در لایه اول یک

```
display = Display(visible=0, size=(1400, 900))
display.start()

def show_video():
  mp4list = glob.glob('video/*.mp4')
  if len(mp4list) > 0:
      mp4 = mp4list[0]
      video = io.open(mp4, 'r+b').read()
      encoded = base64.b64encode(video)
      ipythondisplay.display(HTML(data='''<video alt="test" autoplay loop controls style="height:
    ↪  400px;">
                  <source src="data:video/mp4;base64,{0}" type="video/mp4" />
               </video>'''.format(encoded.decode('ascii'))))
  else:
      print("Could not find video")
```

Code 4: initializing a virtual display

```
Transition = namedtuple('Transition', ('state', 'action', 'next_state', 'reward', 'done'))

class ExperienceReplay():
    def __init__(self, capacity):
        self.memory = deque([], maxlen=capacity)

    def store_trans(self, s, a, sp, r, done):
        transition = Transition(s, a, sp, r, done)
        self.memory.append(transition)

    def sample(self, batch_size):
        return random.sample(self.memory, batch_size)

    def __len__(self):
        return len(self.memory)
```

Code 5: Experience replay

لایه تماما متصل با تابع فعال ساز ReLU استفاده شده است. سپس از Layer normalizaiton استفاده شده تا روند یادگیری را پایدار سازد و سرعت ببخشد، همچنین برای جلوگیری از overfitting از روش Dropout استفاده شده است و نرخ آن ۱۰ درصد است. لایه دوم نیز یک لایه خطی به صورت تماما متصل است و دوباره از نرمالیزه کردن و Dropout استفاده شده و تابع فعال ساز ReLU است. لایه سوم نیز به همین صورت. متد forward ورودی تنسور x را به شبکه می دهد و در خروجی Q value های مربوط به هر اکشن بازگردانده می شود.

```python
class DeepQNetwork(nn.Module):
    def __init__(self, state_size, action_size):
        super(DeepQNetwork, self).__init__()
        self.net = nn.Sequential(
            nn.Linear(state_size, 512),
            nn.ReLU(),
            nn.LayerNorm(512),
            nn.Dropout(0.1),
            nn.Linear(512, 512),
            nn.ReLU(),
            nn.LayerNorm(512),
            nn.Dropout(0.1),
            nn.Linear(512, 512),
            nn.ReLU(),
            nn.Linear(512, action_size)
        )

    def forward(self, x):
        return self.net(x)
```

Code 6: DeepQNetwork class

کلاس بعدی DQNAgent است که Agent مارا پیاده سازی می کند. ایجنتی که با محیط interact می کند، experience هارا ذخیره می کند و Q value هارا بروز رسانی می کند. ابتدا نیاز است تا متغیر هارا در کلاس initialize کنیم. این موارد شامل ابعاد state و action و یا batch size است. همچنین نیاز به تعریف gamma داریم که همان discount factor است. tau پارامتر soft update برای شبکه هدف است. همچنین نیاز به experience replay و یک بهینه ساز و چند مورد دیگر نیز داریم.

متد بعدی در این کلاس take action است که در برنامهٔ ۷ آمده است. که بر اساس epsilon greedy policy از حالت فعلی یک اکشن انتخاب می کند. اگر یک عدد رندوم از eps بزرگتر باشد، ایجنت اکشنی را انتخاب می کند که بیشترین Q-value که در شبکه اصلی پیش بینی شده دارد. در غیر این صورت ایجنت یک اکشن رندوم انتخاب می کند و این تفاوت بین exploitation و exploration است. این روش مطمعن می شود که بالانس بین این دو روش در آموزش رعایت می شود.

متد بعدی update params است که در برنامهٔ ۸ آمده است. این متد پارامتر های شبکه اصلی را به کمک تجربه های گرفته شده از replay buffer بروزرسانی می کند و یک soft update را روی شبکه هدف انجام می دهد. در واقع به کمک sample(batch size) یک بچ از تجربه را از reply buffer برمیدارم و با فرمول بلمن مقدار Q value را حساب

```
    def take_action(self, state, eps=0.0):
        self.value_net.eval()
        if random.random() > eps:
            with torch.no_grad():
                return torch.argmax(self.value_net(torch.tensor(state).float().unsqueeze(0).to(
    ↪ device))).item()
        else:
            return np.random.randint(0, self.action_size)
```

Code 7: take action

میکند. سپس اختلاف بین مقدار پیش بینی شده و اصلی را می یابد و یک گرادیان کاهشی برای مینیمایز کردن این خطا اجرا می کند.

در نهایت به کمک برنامهٔ ۹ این وزن هارا ذخیره و بارگزاری می کند.

حالا باید برای سه batch مختلف، عمل آموزش را انجام دهیم. توجه شود که تمامی مراحل توضیح داده شده و همچنین آموزش برای سه حالت یکسان است و فقط batch size تغییر می کند. برای مثال برای بچ ۳۲، به صورت برنامهٔ ۱۰ خواهد بود.

حال به توضیح فرآیند آموزش می پیردازیم. ابتدا به صورت برنامهٔ ۱۱ ایجنت را تعریف می کنیم و همچنین پاداش تجمعی را initialize می کنیم. در این قسمت ما یک ایجنت از DQNAgent با حالت و اکشن خاص درست می کنیم که بچ سایز اش را مشخص کرده ایم. همچنین صف های ۲۵ تایی تشکیل می دهیم که در آینده از آنها میانگین می گیریم.

در برنامهٔ ۱۲ حلقه آموزش را به تعداد مشخصی از اپیزود که در این حالت ۲۵۰ است تشکیل می دهیم. در این کد از هر ۵۰ امین اپیزود فیلم برداری می شود یعنی اپیزود ۵۰ ام ۱۰۰ ام ۱۵۰ ام ۲۰۰ ام و ۲۵۰ ام. سپس محیط را ریست می کند و مقادیر اولیه را میگیرد. سپس done را که قبل تر توضیح داده شدذه بود به حالت false در می آورد تا اپیزود بعدی استارت شود. سپس مقدار پاداش برای اپیزود فعلی را صفر می گذارد.

بخش برنامهٔ ۱۳ تا زمانی که اپیزود تمام شود با محیط ارتباط می گیرد، تجربه کسب می کند و Q value هارا به روز می کند. در واقع ابتدا یک اکشن انتخاب می شود، حالت بعدی، فلگ و پاداش را میگیرد و تجربه کسب شده را در reply buffer ذخیره می کند. سپس Q value function را به روز رسانی می کند (به کمک سمپل گرفته از reply buffer) سپس حالت را زا حالت فعلی به حالت بعدی تغییر میدهد و پاداش را به پاداش های قبلی در آن اپیزود اضافه میکند.

بخش بعدی اپسیلون را کاهش می دهد و مقدار پاداش تجمعی در این اپیزود را ذخیره می کند. اگر اپیزود مضربی از ۵۰ بود آن را ذخیره می کند و اگر مضرب ۲۵ بود، پاداش میانگین آم و اپیزود فعلی و مقدار اپسیلون را پرینت می کند. این بخش در برنامهٔ ۱۴ آمده است.

در نهایت پاداش تجمعی برای batch size = 32 در شکل ۳ آمده است. همین نمودار برای batch size = 64 در شکل ۴ آمده است. که همانگونه که دیده میشود وضعیت پایدار تری را دارد ولی در انتها پاداش batch size = 32 بهتر است و نشان می دهد ایجنت بهتر عمل کرده است.

همین نمودار برای batch size = 128 در شکل ۵ آمده است. که همانگونه که دیده میشود وضعیت ناپایدار تری را دارد . مطابق شکل ۳ تا شکل ۵ در مراحل اولیه مقدار پاداش نوسان زیادی دارد که نشان دهنده فاز exploration ایجنت

```python
def update_params(self):
        if len(self.experience_replay) < self.batch_size:
            return
        batch = Transition(*zip(*self.experience_replay.sample(self.batch_size)))

        state_batch = torch.tensor(np.array(batch.state), dtype=torch.float32).to(device)
        action_batch = torch.tensor(batch.action, dtype=torch.int64).unsqueeze(1).to(device)
        next_state_batch = torch.tensor(np.array(batch.next_state), dtype=torch.float32).to(
    ↪ device)
        reward_batch = torch.tensor(batch.reward, dtype=torch.float32).unsqueeze(1).to(device)
        done_batch = torch.tensor(batch.done, dtype=torch.float32).unsqueeze(1).to(device)

        q_expected = self.value_net(state_batch).gather(1, action_batch)
        q_targets_next = self.target_net(next_state_batch).detach().max(1)[0].unsqueeze(1)
        q_targets = reward_batch + (self.gamma * q_targets_next * (1 - done_batch))
        loss = F.mse_loss(q_expected, q_targets)

        self.optimizer.zero_grad()
        loss.backward()
        self.optimizer.step()

        # Soft update target network parameters
        for target_param, local_param in zip(self.target_net.parameters(), self.value_net.
    ↪ parameters()):
            target_param.data.copy_(self.tau * local_param.data + (1.0 - self.tau) * target_param
    ↪ .data)
```

Code 8: update params

```python
    def save(self, fname):
        torch.save(self.value_net.state_dict(), fname)

    def load(self, fname):
        self.value_net.load_state_dict(torch.load(fname, map_location=device))
```

Code 9: save and load the weights

```python
# NOTE: DON'T change values
n_episodes = 250
eps = 1.0
eps_decay_rate = 0.97
eps_end = 0.01
BATCH_SIZE = 32
```

Code 10: batch size 32

```
agent = DQNAgent(state_size, action_size, batch_size=BATCH_SIZE)
crs = np.zeros(n_episodes)
crs_recent = deque(maxlen=25)
```

Code 11: initialize agent

```
for i_episode in range(1, n_episodes + 1):
    env = RecordVideo(gym.make("LunarLander-v2", render_mode="rgb_array"), f"./DQN/batch{
    ↪ BATCH_SIZE}/eps{i_episode}") if i_episode  50 == 0 else gym.make("LunarLander-v2",
    ↪ render_mode="rgb_array")
    state, info = env.reset()
    done = False
    cr = 0
```

Code 12: train loop

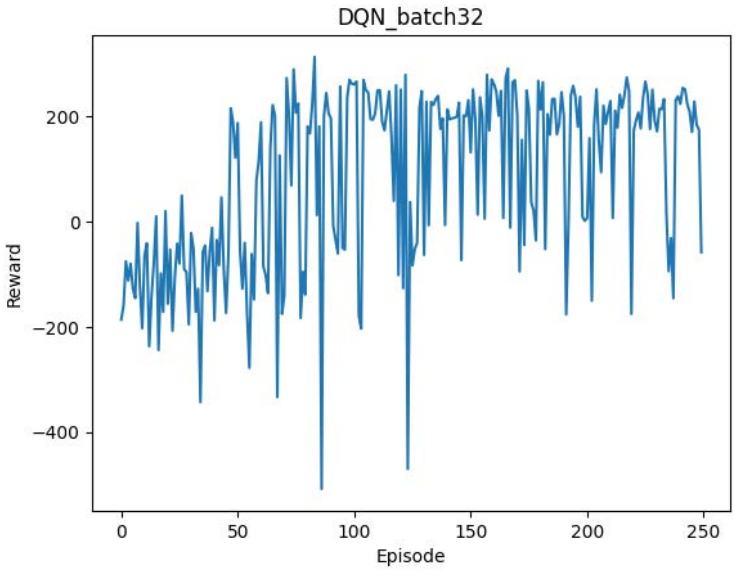
```
    while not done:
        action = agent.take_action(state, eps)
        next_state, reward, done, truncated, info = env.step(action)
        agent.experience_replay.store_trans(state, action, next_state, reward, done or truncated)
        agent.update_params()
        state = next_state
        cr += reward
```

Code 13: episode loop

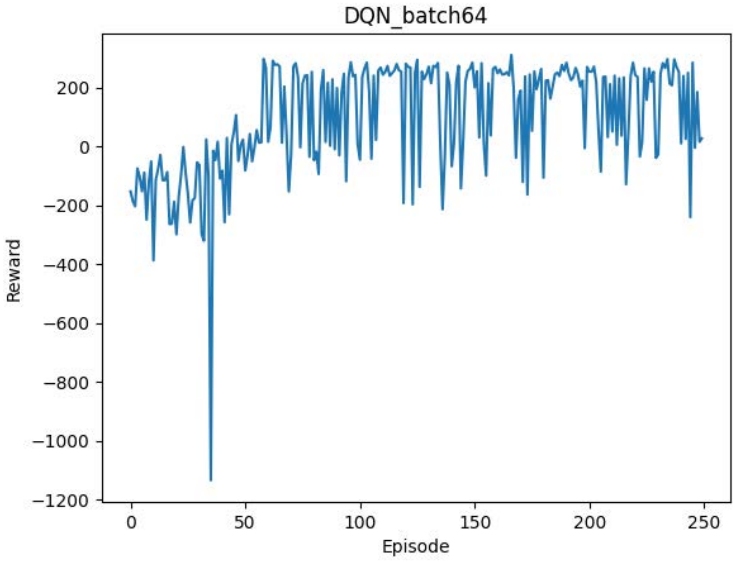
```
    eps = max(eps * eps_decay_rate, eps_end)
    crs[i_episode - 1] = cr
    crs_recent.append(cr)
    if i_episode % 50 == 0:
        agent.save(f"q_net_batch{BATCH_SIZE}_eps{i_episode}.pt")

    print(f'\rEpisode {i_episode}\tAverage Reward: {np.mean(crs_recent):.2f}\tEpsilon: {eps:.2f}'
    ↪ , end="")
    if i_episode % 25 == 0:
        print(f'\rEpisode {i_episode}\tAverage Reward: {np.mean(crs_recent):.2f}\tEpsilon: {eps
    ↪ :.2f}')
```

Code 14: Epsilon Decay



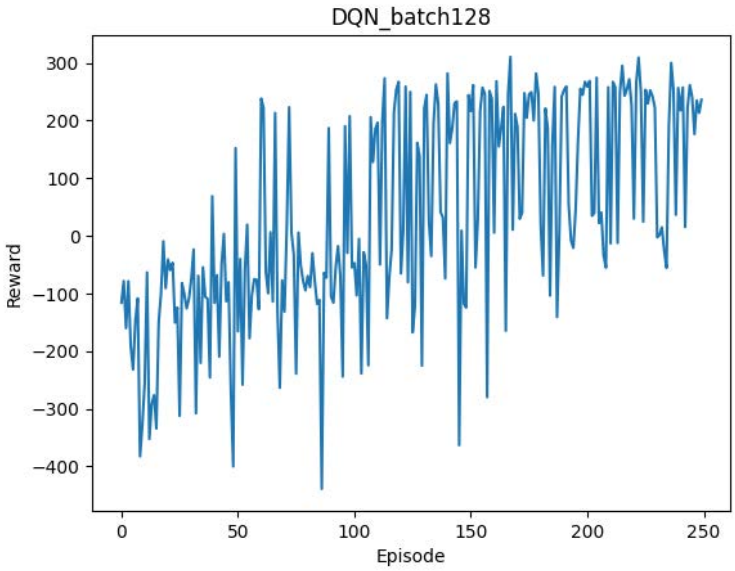
شکل ۳: نمودار پاداش تجمعی برای batch size = 32



شکل ۴: نمودار پاداش تجمعی برای batch size = 64

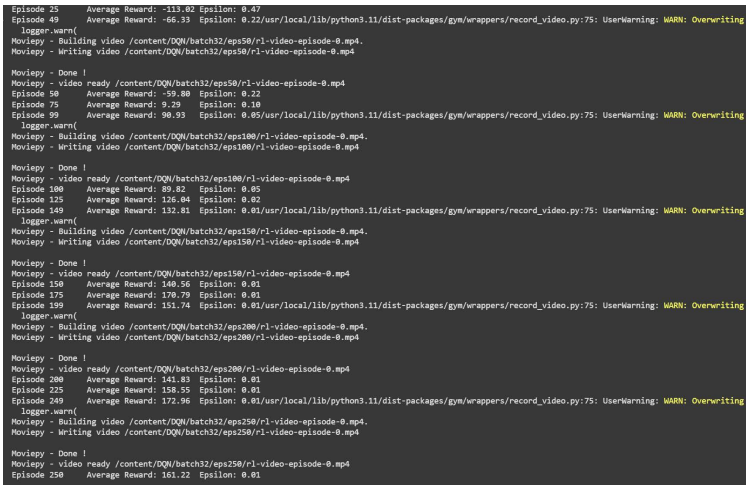
است.

مطابق شکل ۴ و batch size = 64 در حدود اپیزود ۲۰ یک پاداش منفی بسیار زیاد میگیرد که نشان دهنده یک حادثه خیلی بد برای فضاپیما است. بعد از اپیزود ۵۰ دیده می شود که پاداش ها شروع به صعودی شدن می کنند و البته کمی نوسانات وجود دارد ولی با جلو رفتن بهتر وضعیت بهتر می شود. از اپیزود ۱۰۰ پاداش ها حدود ۱۰۰ یا مثبت هستند که نشان می دهد ایجنت در حال پایدارسازی عملکرد و رفتار خود است. می توان گفت در هر سه حالت عملکرد ایجنت با مرور زمان و زیادتر شدن تعداد episode بهبود می یابد و نشان می دهد که الگوریتم درست کار می کند.



شکل ۵: نمودار پاداش تجمعی برای batch size = 128

می توان میانگین هر ۲۵ اپیزود را برای بچ های متفاوت در شکل ۶ تا شکل ۸ مشاهده کرد. مطابق این نمودار ها و به صورت شهودی و با توجه به شکل ۳ تا شکل ۵ می توان گفت که بهترین عملکرد مدل مربوط به حالت batch size = 128 است. این حالت بهترین بالانس را بین دو فاز exploraiton و exploatation ارائه می دهدکه در نهایت به بیشترین پاداش ها منجر می شود. در جایگاه ها بعدی به ترتیب batch size = 64 و batch size = 32 قرار دارند که روند بهبود پاداش در آن ها با سرعت کمتری دنبال می شود.



شکل ۶: پاداش تجمعی برای batch size = 32

حال به سراغ ارزیابی این سه حالت به کمک متریک regret می رویم. ابتدا تعریفی از این معیار را با هم مرور کنیم.

این معیار برای اندازی گیری اوضاع در شرایط تصمیم گیری و در فضای یاگیری تقویتی به کار می رود. این معیار

```
Episode 25      Average Reward: -147.19 Epsilon: 0.47
Episode 49      Average Reward: -135.71 Epsilon: 0.22Moviepy - Building video /content/DQN/batch64/eps50/rl-video-episode-0.mp4.
Moviepy - Writing video /content/DQN/batch64/eps50/rl-video-episode-0.mp4

Moviepy - Done !
Moviepy - video ready /content/DQN/batch64/eps50/rl-video-episode-0.mp4
Episode 50      Average Reward: -131.23 Epsilon: 0.22
Episode 75      Average Reward: 103.58  Epsilon: 0.10
Episode 99      Average Reward: 125.47  Epsilon: 0.05Moviepy - Building video /content/DQN/batch64/eps100/rl-video-episode-0.mp4.
Moviepy - Writing video /content/DQN/batch64/eps100/rl-video-episode-0.mp4

Moviepy - Done !
Moviepy - video ready /content/DQN/batch64/eps100/rl-video-episode-0.mp4
Episode 100     Average Reward: 125.91  Epsilon: 0.05
Episode 125     Average Reward: 186.64  Epsilon: 0.02
Episode 149     Average Reward: 149.47  Epsilon: 0.01Moviepy - Building video /content/DQN/batch64/eps150/rl-video-episode-0.mp4.
Moviepy - Writing video /content/DQN/batch64/eps150/rl-video-episode-0.mp4

Moviepy - Done !
Moviepy - video ready /content/DQN/batch64/eps150/rl-video-episode-0.mp4
Episode 150     Average Reward: 150.82  Epsilon: 0.01
Episode 175     Average Reward: 159.18  Epsilon: 0.01
Episode 199     Average Reward: 205.65  Epsilon: 0.01Moviepy - Building video /content/DQN/batch64/eps200/rl-video-episode-0.mp4.
Moviepy - Writing video /content/DQN/batch64/eps200/rl-video-episode-0.mp4

Moviepy - Done !
Moviepy - video ready /content/DQN/batch64/eps200/rl-video-episode-0.mp4
Episode 200     Average Reward: 206.68  Epsilon: 0.01
Episode 225     Average Reward: 145.76  Epsilon: 0.01
Episode 249     Average Reward: 167.85  Epsilon: 0.01Moviepy - Building video /content/DQN/batch64/eps250/rl-video-episode-0.mp4.
Moviepy - Writing video /content/DQN/batch64/eps250/rl-video-episode-0.mp4

Moviepy - Done !
Moviepy - video ready /content/DQN/batch64/eps250/rl-video-episode-0.mp4
Episode 250     Average Reward: 158.33  Epsilon: 0.01
```

شکل ۷: پاداش تجمعی برای batch size = 64

```
Episode 25      Average Reward: -166.92 Epsilon: 0.47
Episode 49      Average Reward: -132.26 Epsilon: 0.22Moviepy - Building video /content/DQN/batch128/eps50/rl-video-episode-0.mp4.
Moviepy - Writing video /content/DQN/batch128/eps50/rl-video-episode-0.mp4

Moviepy - Done !
Moviepy - video ready /content/DQN/batch128/eps50/rl-video-episode-0.mp4
Episode 50      Average Reward: -121.19 Epsilon: 0.22
Episode 75      Average Reward: -42.09  Epsilon: 0.10
Episode 99      Average Reward: -64.74  Epsilon: 0.05Moviepy - Building video /content/DQN/batch128/eps100/rl-video-episode-0.mp4.
Moviepy - Writing video /content/DQN/batch128/eps100/rl-video-episode-0.mp4

Moviepy - Done !
Moviepy - video ready /content/DQN/batch128/eps100/rl-video-episode-0.mp4
Episode 100     Average Reward: -65.64  Epsilon: 0.05
Episode 125     Average Reward: 52.43   Epsilon: 0.02
Episode 149     Average Reward: 66.80   Epsilon: 0.01Moviepy - Building video /content/DQN/batch128/eps150/rl-video-episode-0.mp4.
Moviepy - Writing video /content/DQN/batch128/eps150/rl-video-episode-0.mp4

Moviepy - Done !
Moviepy - video ready /content/DQN/batch128/eps150/rl-video-episode-0.mp4
Episode 150     Average Reward: 66.59   Epsilon: 0.01
Episode 175     Average Reward: 141.83  Epsilon: 0.01
Episode 199     Average Reward: 138.29  Epsilon: 0.01Moviepy - Building video /content/DQN/batch128/eps200/rl-video-episode-0.mp4.
Moviepy - Writing video /content/DQN/batch128/eps200/rl-video-episode-0.mp4

Moviepy - Done !
Moviepy - video ready /content/DQN/batch128/eps200/rl-video-episode-0.mp4
Episode 200     Average Reward: 140.79  Epsilon: 0.01
Episode 225     Average Reward: 160.91  Epsilon: 0.01
Episode 249     Average Reward: 161.18  Epsilon: 0.01Moviepy - Building video /content/DQN/batch128/eps250/rl-video-episode-0.mp4.
Moviepy - Writing video /content/DQN/batch128/eps250/rl-video-episode-0.mp4

Moviepy - Done !
Moviepy - video ready /content/DQN/batch128/eps250/rl-video-episode-0.mp4
Episode 250     Average Reward: 169.64  Epsilon: 0.01
```

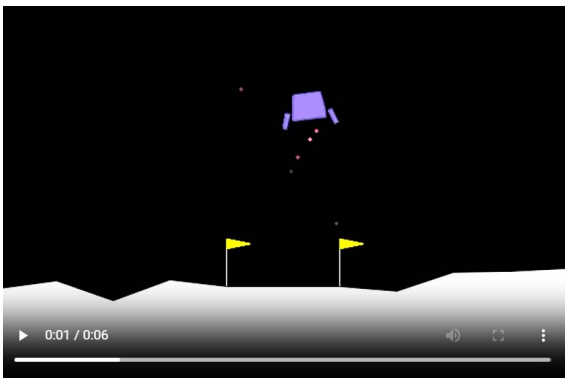
شکل ۸: پاداش تجمعی برای batch size = 128

تفاوت بین پاداشی که توسط ایجنت دریافت شده و پاداشی که دریافت می شد، اگر ایجنت بهترین تصمیم را در هر اپیزود میگرفت ارزیابی می کند. پس می توان آن را به صورت رابطهٔ ۱ فرموله کرد.

$$R_t = \mu^\star - \mu_a \qquad (۱)$$

در رابطهٔ ۱ $\mu^\star$ پاداشی است که اگر ایجنت بهترین اکشن را انتخاب می کرد دریافت می کرد. در طرف دیگر μ_a پاداشی است که ایجنت با انتخاب کردن اکشن a در زمان t آن را دریافت کرده است.
حال اگر ما رابطهٔ ۱ را در تمامی اپیزود ها حساب کنیم می توانیم regret تجمعی برای ایجنت را بدست آوریم.
در واقع می توان گفت regret هزینه انتخاب نکردن بهترین تصمیم در هر اپیزود را به ما نشان می دهد.
مطابق نتایج حالت batch size = 32 در ابتدا regret متوسط است، سپس این معیار به صورت واضحی کم میشود چون reward ها د رحال افزایش هستند (حدود اپیزود ۵۰) و در نهایت در اپیزود های بالاتر این معیار به کمترین حالت خود میرسد از آنجایی که پاداش ها در حال افزایش هستند.
برای حالت batch size = 128 ما کمترین میزان regret را در فاز های مختلف شاهد هستیم چون بیشتری پاداش هارا برای این حالت دریافت می کنیم و می توان اینطور توصیف کرد که در این حالت ایجنت به بهترین عملکرد خود نزدیک است. پس با توجه به این معیار نیز بهترین گزینه ما حالت batch size = 128 است.

در نهایت این بخش به سراغ تهیه فیلم ها می رویم که در روند آموزش آن هارا ذخیره کرده ایم. توجه شود که تمامی فیلم های مربوط به هر سه بچ در فایل ارائه آمده است.
برای حالت batch size = 128 این ویدیو ها روی فایل google collab به صورت شکل ۹ قرار گرفته اند.



شکل ۹: ویدیویی از ایجنت در محیط به ازای batch size = 128 در اپیزود ۲۵۰

Episode 250

```
import io
import base64
from IPython.display import HTML
from IPython.display import display as ipythondisplay

def show_video(video_path):
    with open(video_path, 'rb') as f:
        video = f.read()
    encoded = base64.b64encode(video).decode('ascii')
    ipythondisplay(HTML(data=f'''
        <video alt="RL Video" autoplay loop controls style="height: 400px;">
            <source src="data:video/mp4;base64,{encoded}" type="video/mp4" />
        </video>'''))

# Example usage:
video_path = "/content/DQN/batch128/eps250/rl-video-episode-0.mp4"
show_video(video_path)
```

شکل:۱۰ کد نمایش ویدیو ایجنت در محیط batch size= 128 در اپیزود ۲۵۰ از حافظه کولب

این ویدیو ها به ازای تمامی اپیزود ها در فایل google collab قرار گرفته اند.

قسمت سوم

باید در این قسمت مدل DDQN را به جای مدل DQN جاگذاری کنیم. می توان گفت تفاوت اصلی این دو مدل در نحوه بروز کرد Q value ها و شبکه هدف می باشد. به طور دقیق تر DQN از شبکه استفاده می کند تا Q value هارا تخمین بزند اما DDQN از شبکه موجود اکشن را انتخاب می کند و Q value های مرحله بعد را به کمک شبکه هدف تخمین می زند. این تفاوت در برنامهٔ ۱۵ و برنامهٔ ۱۶ قابل مشاهده است.

```
q_targets_next = self.target_net(next_state_batch).detach().max(1)[0].unsqueeze(1)
q_targets = reward_batch + (self.gamma * q_targets_next * (1 - done_batch))
```

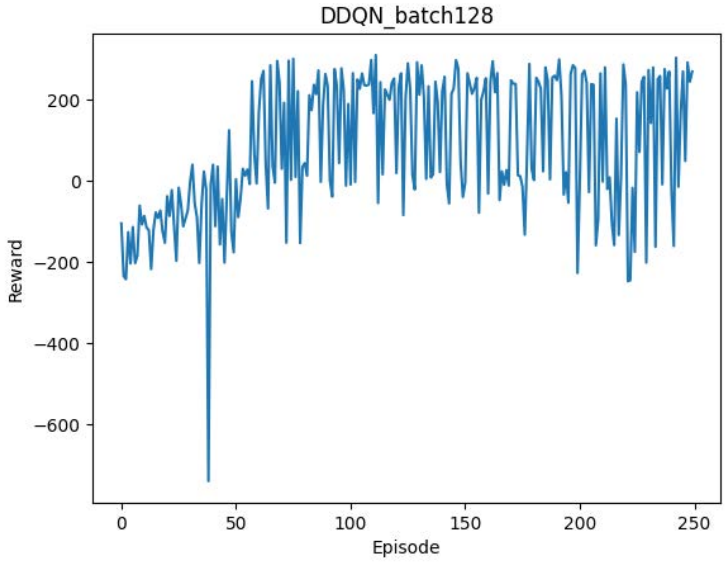
Code 15: DQN Agent

```
next_action_batch = torch.argmax(self.value_net(next_state_batch), dim=1, keepdim=True)
q_targets_next = self.target_net(next_state_batch).gather(1, next_action_batch).detach()
q_targets = reward_batch + (self.gamma * q_targets_next * (1 - done_batch))
```

Code 16: DDQN Agent

مشاهده می شود که DDQN یک next action batch دارد که انتخاب اکشن با کمک شبکه فعلی را انجام می دهد که در DQN وجود ندارد. خوبی این روش این است که از بایاس های احتمالی در شبکه DQN جلوگیری می کند چون گام های انتخاب اکشن و تخمین Q value با هم تفاوت دارد. در نهایت این روش می تواند به پاسخ هایی پایدارتر و با اتکاتری را ارائه دهد.

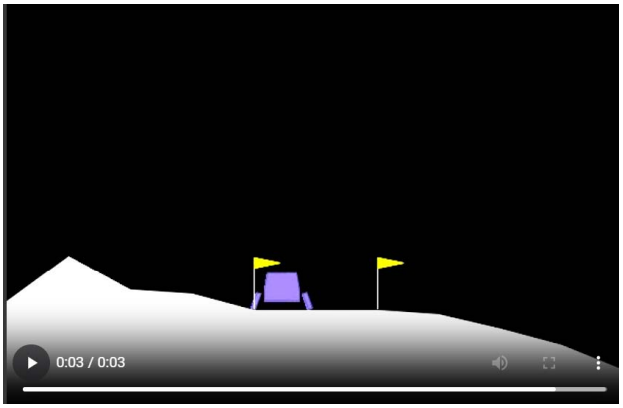
نتایج شبیه سازی با کمک DDQN در حالت batch size = 128 در شکل ۱۱ آمده است. در این حالت دیده می شود



شکل ۱۱: نمودار پاداش تجمعی به کمک DDQN در حالت batch size = 128

که پاداش ها بسیار پایدار تر از حالت قبل هستند، یعنی پایداری ایجنت در دریافت پاداش های بیشتر بهتر شده است.

برای تهیه ویدیو مانند بخش قبل از دستورات مربوط به نمایش ویدیو در محیط google collab استفاده می کنیم و به صورت گفته شده برای دو حالت اپیزود ۱۰۰ و ۲۵۰ ویدیو را نشان می دهیم. قابل ذکر است این ویدیو ها در فایل ارسالی قرار دارند.



شکل ۱۲: ویدیویی از ایجنت در محیط به ازای batch size = 128 در اپیزود ۲۵۰