دانشکده مهندسی برق

دوره کارشناسی ارشد مهندسی برق-کنترل

مینی‌پروژه چهارم درس یادگیری ماشین

توسط:

محمدرضا امانی – احمدرضا طاهری

استاد راهنما:

دکتر مهدی علیاری شوره‌دلی

بهار ۱۴۰۴


بسم الله الرحمن الرحیم

فهرست مطالب

فهرست شکل ها

سوال ۱

الف) طراحی شبکه با استفاده از یک نورون ساده یاMcCulloch-Pitts

در این بخش، هدف طراحی یک نورون ساده از نوع پرسپترون یا McCulloch-Pitts است که قادر باشد ناحیه‌ی هاشورزده‌ی داخل مثلث مشخص‌شده در نمودار شکل ۱(آ) را از سایر نواحی اطراف تفکیک کند.
نورون McCulloch-Pitts یکی از ابتدایی‌ترین مدل‌های نورون مصنوعی است که بر پایه جمع وزن‌دار ورودی‌ها و مقایسه آن با یک آستانه (threshold) عمل می‌کند. تابع فعال‌ساز به‌کاررفته در این مدل، (step function) یا همان تابع پله است که خروجی آن فقط دو مقدار ۰ و ۱ را تولید می‌کند.

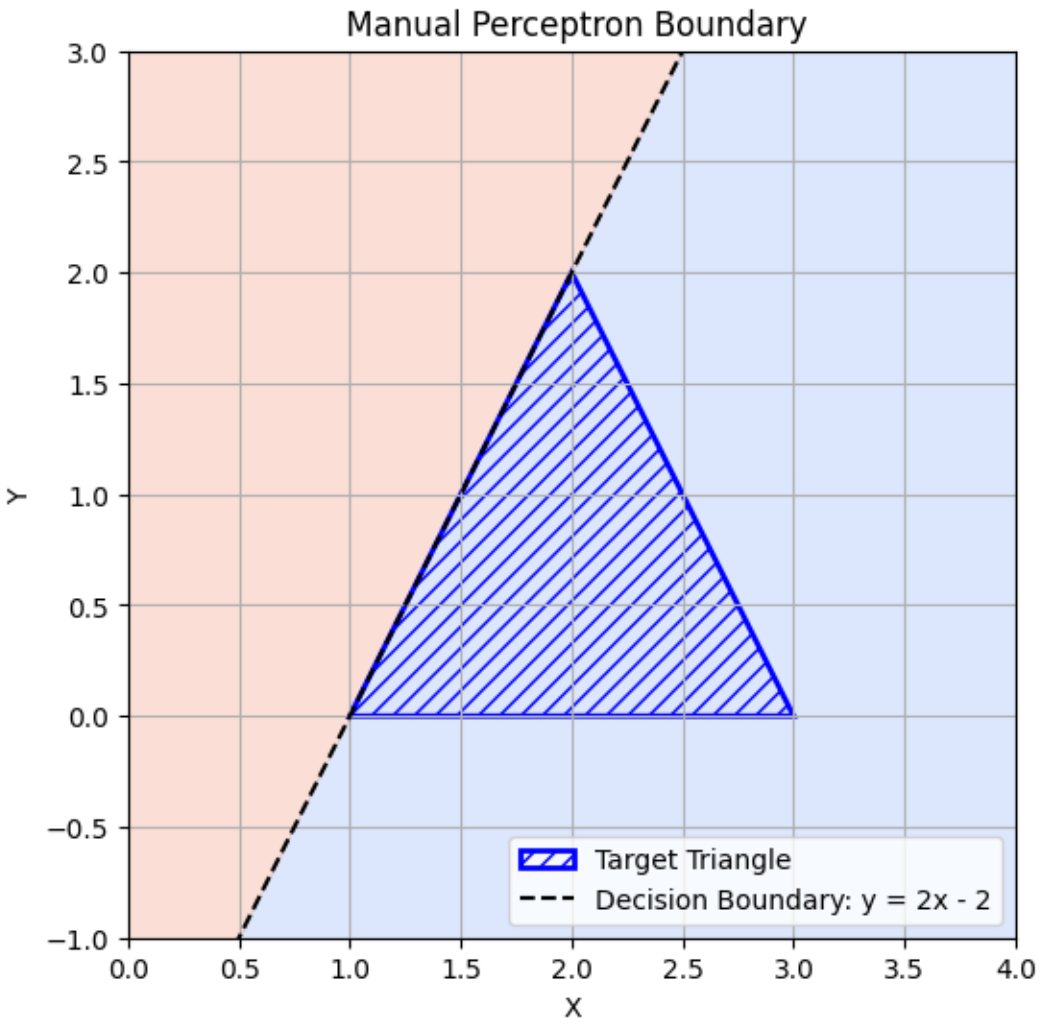
$$f(z) = \begin{cases} 1, & z \geq 0 \\ 0, & z < 0 \end{cases}$$

با این‌وجود، یک چالش مهم در طراحی این مدل، عدم امکان جداسازی دقیق ناحیه مثلثی با استفاده از تنها یک نورون ساده است. ناحیه مورد نظر یک ناحیه چندضلعی (مثلثی) است که برای جداسازی دقیق آن به چندین مرز غیرموازی نیاز است. از آنجایی که یک نورون ساده تنها قادر به تولید یک مرز خطی در فضای دوبعدی است، لذا فقط می‌تواند ناحیه مثلثی را به‌صورت تقریبی از سایر نواحی تفکیک کند. در نتیجه، تفکیک کامل فقط با یک نورون امکان‌پذیر نیست و خطا اجتناب‌ناپذیر خواهد بود.
در این پروژه، برای دستیابی به بهترین تقریب ممکن با یک نورون، یکی از اضلاع مثلث به‌عنوان مرز تصمیم در نظر گرفته شد. ضلع بین نقاط C(1, 0) وA(2, 2) ، دارای معادله خطی زیر است:

$$y = 2x - 2 \ \rightarrow z = y - 2x + 2$$

بنابراین بردار وزن را به صورت w = [-2,1] و بایاس b = -2 انتخاب می شود.

خروجی مرز تصمیم گیری به صورت زیر خواهد بود:



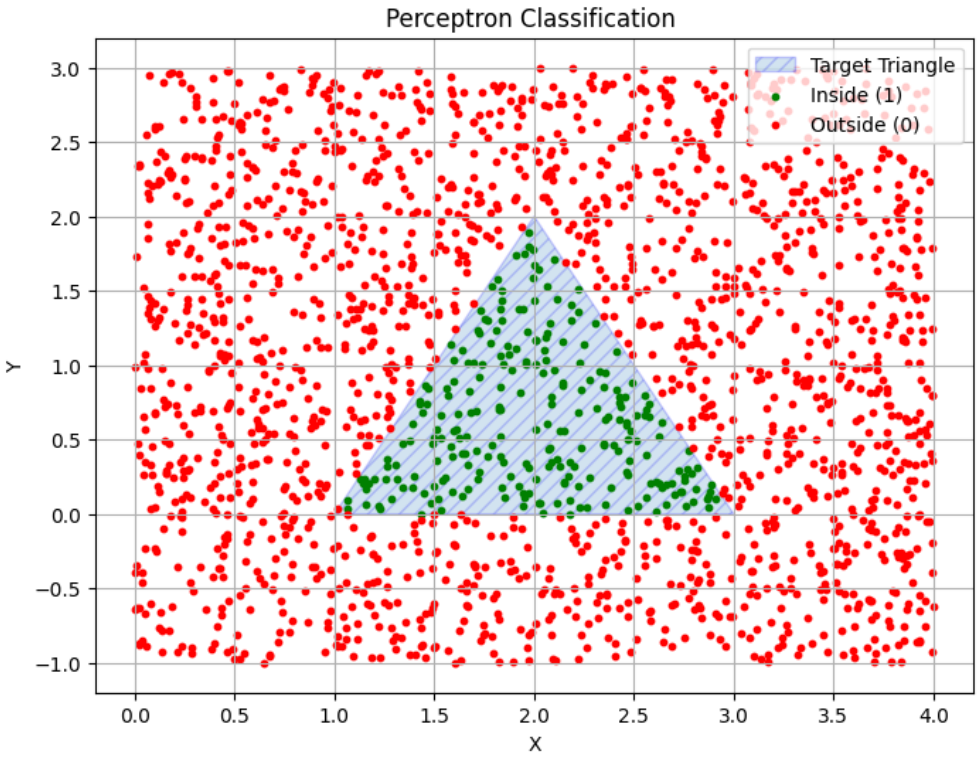
شکل ۱_۱ مرز تصمیم گیری شبکه طراحی شده دستی

در ادامه برای جداسازی ناحیه داخل مثلث از سایر نواحی از الگوریتم پرسپترون ساده (Single-layer Perceptron) استفاده خواهیم کرد که مدلی پایه‌ای در یادگیری ماشین که تنها شامل یک نورون با وزن‌ها و بایاس قابل تنظیم و یک تابع فعال‌ساز پله‌ای می‌باشد.

به منظور ایجاد مجموعه داده برای آموزش مدل، تعداد ۲۰۰۰ نقطه تصادفی با توزیع یکنواخت در بازه‌های مجاز تولید میکنیم و با استفاده از تابع contains_points از کتابخانه‌ی matplotlib.path لیبل گذاری داخل و خارج مثلث را انجام میدهیم.

پس از تولید داده و برچسب‌گذاری آن‌ها، فرآیند آموزش مدل آغاز می شود. شبکه مورد استفاده از یک نورون با دو ورودی (مختصات x و y)تشکیل شده و از تابع فعال‌ساز پله‌ای برای تصمیم‌گیری استفاده می‌کرد. وزن‌های اولیه و بایاس به‌صورت تصادفی مقداردهی شده و آموزش مدل در طی ۳۰ تکرار (epoch) بر کل داده‌ها انجام گرفت. در هر تکرار، مدل با استفاده از قانون یادگیری پرسپترون، وزن‌ها و بایاس را بر اساس خطای بین خروجی پیش‌بینی‌شده و مقدار واقعی به‌روزرسانی می‌کرد. به‌طور مشخص، اگر خروجی مدل با برچسب واقعی مطابقت نداشت، وزن‌ها متناسب با بردار ورودی و نرخ یادگیری اصلاح می‌شدند.

نتیجه آموزش شبکه و کلاس بندی به صورت زیر می باشد :



شکل ۲_۱ نتیجه کلاسبندی با پرسپترون ساده

همانطور که مشاهده می شود، مدل قادر به تشخیص حدودی مرز جدایی بین نقاط داخل و خارج مثلث شد. این مرز، یک خط مستقیم در صفحه است که توسط معادله زیر بیان می شود:

```
Decision boundary equation:
   0.1641 * x + 0.0587 * y + -0.0523 = 0
⟹    y = -0.1641/(0.0587) * x + 0.0523/(0.0587)
```

شکل ۳_۱ معادله خط تصمیم گیری پرسپترون ساده

ب.۱) توسعه شبکه طراحی شده برای کلاسبندی نقاط داخل و خارج مثلث

در دفترچه تمرین ارائه‌شده، نمونه کدی برای تشخیص ناحیه‌ای مستطیلی با استفاده از نورون‌های مدل McCulloch-Pitts در اختیار قرار گرفته بود. در این کد، با تعریف چهار نورون اولیه که هرکدام یک مرز از مستطیل را مدل می‌کردند، و یک نورون نهایی که نقش تجمیع‌کننده با عملگر AND را داشت، نقاط درون مستطیل از نقاط خارج آن تفکیک شده بودند. خروجی نهایی این شبکه به این صورت بود که در صورت فعال شدن هر چهار نورون اولیه (یعنی قرار گرفتن نقطه در محدوده مستطیل)، خروجی برابر با یک و در غیر این صورت برابر با صفر بود.

در تمرین حاضر، هدف جایگزینی ناحیه مستطیلی با ناحیه‌ای مثلثی با رأس‌های مشخص A(1,0)، B(3,0) و C(2,2) بود. برای این منظور، باید سه نورون اولیه طراحی می‌شدند که هرکدام یکی از اضلاع مثلث را مدل کنند. معادلات هر ضلع به‌صورت زیر تعریف شدند:

ضلع AC : معادله مرز به‌صورت $y = 2x - 2$، که برای نقاط پایین‌تر از این خط باید برقرار باشد.

$w = [2,-1]$, $b = 2$

ضلع BC : معادله مرز به‌صورت $y = -2x + 6$، که برای نقاط پایین‌تر از این خط نیز باید برقرار باشد.

$w = [-2,-1]$, $b = -6$

ضلع AB : مرز افقی $y = 0$ که نقاط بالاتر از این مرز مجازند.

$w = [0,1]$, $b = 0$

هر نورون بررسی می‌کند که نقطه ورودی در کدام طرف خط مرزی قرار دارد. سپس یک نورون نهایی با وزن [1, 1, 1] و آستانه 3 به‌صورت عملکرد منطقی AND عمل می‌کند و مشخص می‌سازد که آیا نقطه درون هر سه شرط قرار دارد یا خیر. به عبارت دیگر:

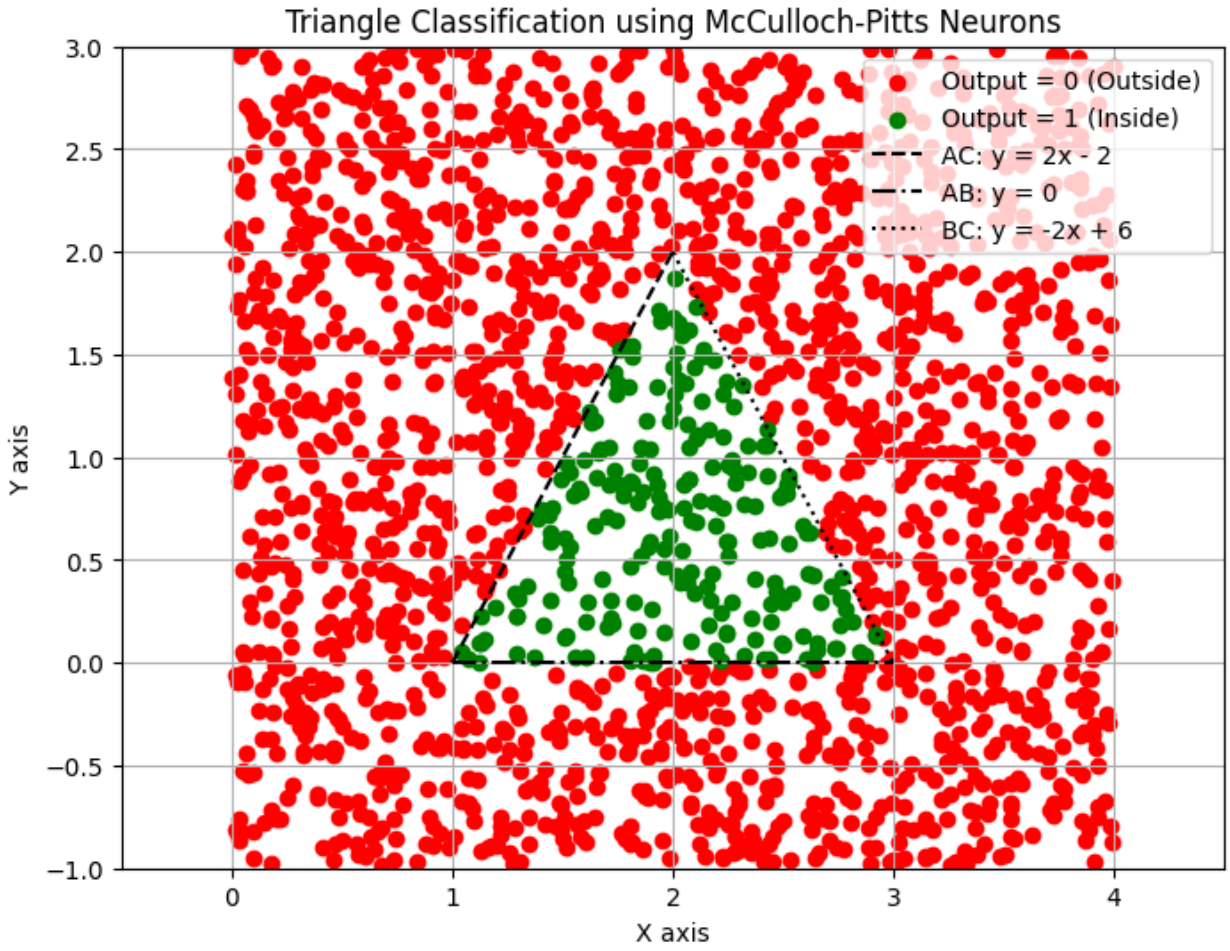
اگر سه نورون اول فعال شوند (به‌معنای آنکه نقطه درون مثلث قرار دارد)، نورون نهایی خروجی 1 تولید می‌کند. در غیر این صورت، خروجی نهایی برابر با 0 خواهد بود.

در ادامه، تعداد ۲۰۰۰ نقطه تصادفی در محدوده‌ی $x \in [0, 4]$ و $y \in [-1, 3]$ تولید شده و هرکدام از آن‌ها به تابع Area(x, y) داده شدند. خروجی این تابع، وضعیت قرارگیری نقطه در درون یا بیرون مثلث را مشخص می‌کرد:

اگر خروجی برابر با 1 بود، نقطه به‌عنوان داخل مثلث در نظر گرفته شده و با رنگ سبز نمایش داده شد.

در غیر این صورت، نقطه به‌عنوان خارج مثلث در نظر گرفته شد و با رنگ قرمز نمایش داده شد.

نتیجه توسعه شبکه طراحی شده به صورت زیر خواهد بود.



شکل ۴_۱ کلاسبندی نقاط داخل و خارج مثلث

ب.۲) بررسی اثر تابع فعالساز Tanh

در این مرحله از پیاده‌سازی شبکه‌ی نورونی McCulloch-Pitts برای تشخیص ناحیه درون مثلث، هدف بررسی تأثیر انتخاب تابع فعال‌ساز پیوسته (tanh) و نحوه ترکیب خروجی سه نورون اولیه برای تصمیم‌گیری نهایی است.

روش اولیه: استفاده از حاصل‌ضرب خروجی نورون‌ها

در ابتدا برای ترکیب خروجی سه نورون مدل‌کننده‌ی اضلاع مثلث، از روش حاصل‌ضرب خروجی‌های آن‌ها استفاده شد. در این حالت، هر نورون با استفاده از تابع فعال‌ساز tanh مقدار خروجی‌ای بین ۱- و ۱+ تولید می‌کرد و سپس خروجی نهایی به صورت زیر محاسبه شد:

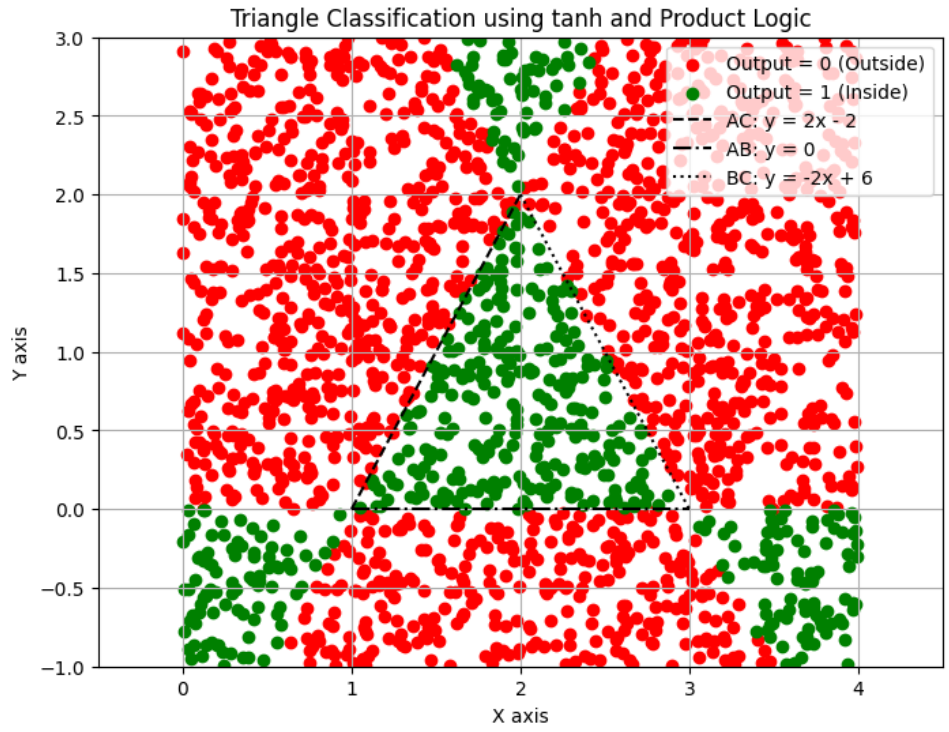
z_final = tanh(z1 * z2 * z3)

در نهایت، این مقدار با صفر مقایسه می‌شد:

اگر z_final > 0 در نظر گرفته می‌شد که نقطه درون مثلث است (رنگ سبز)

در غیر این صورت نقطه خارج از مثلث تلقی می‌شد (رنگ قرمز)

نتیجه چنین پیاده سازی به صورت زیر خواهد بود:



شکل ۵_۱ پیاده سازی منطق ضرب باتابع فعالساز tanh

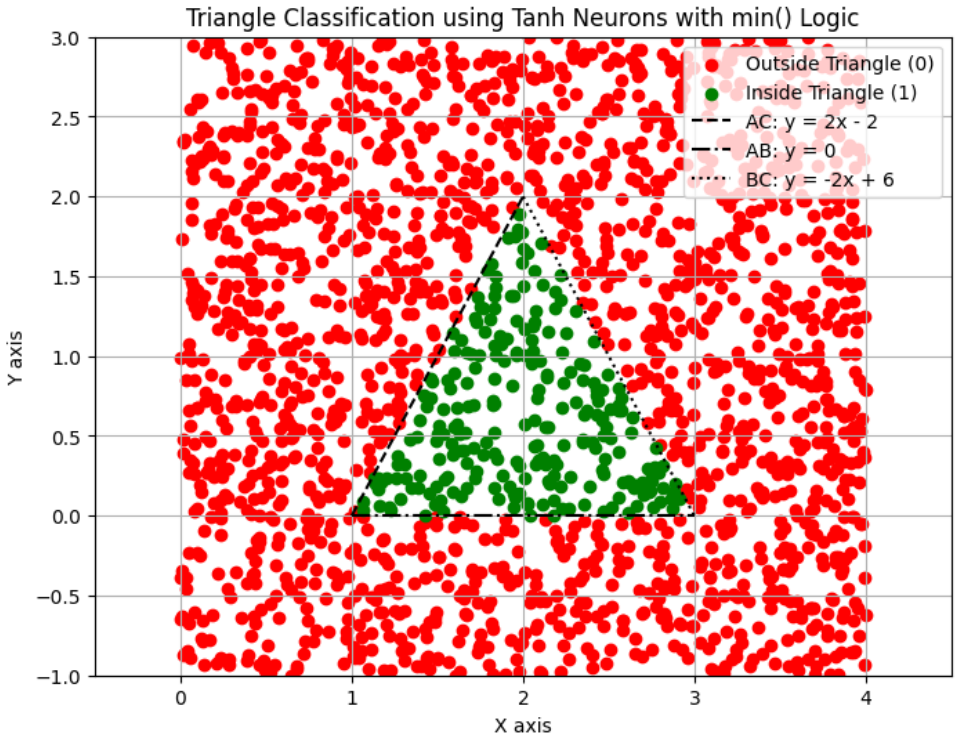
همانطور که مشاهده میشود کلاس بندی به درستی صورت نگرفته که به این علت است که در شرایطی که دو نورون مقدار منفی داشته باشند و یک نورون مقدار مثبت داشته باشد، حاصل‌ضرب مثبت می‌شود و در نتیجه نقطه‌ای که در واقع خارج از مثلث است، به اشتباه به عنوان نقطه داخل (سبز) دسته‌بندی می‌شود. این موضوع باعث اشتباه در مرز تصمیم‌گیری می‌شود و دقت تفکیک ناحیه کاهش می‌یابد.

برای رفع مشکل مذکور، روش بهتری برای ترکیب خروجی‌های سه نورون پیشنهاد می شود. در این روش از تابع min برای انتخاب حداقل مقدار بین سه خروجی استفاده می‌شود به این صورت که :

z_final = min(z1, z2, z3)

این روش دقیقاً منطق AND را در فضای پیوسته شبیه‌سازی می‌کند. اگر حتی یکی از نورون‌ها مقدار منفی بدهد (یعنی یکی از شرط‌های درون مثلث بودن برقرار نباشد)، کل تصمیم به سمت خارج از مثلث تغییر می‌کند. این ویژگی باعث افزایش دقت در تشخیص مرز واقعی ناحیه مثلث می‌شود.

نتیجه چنین پیاده سازی به صورت زیر می باشد :



شکل ۶_۱ پیاده سازی منطق min باتابع فعالساز tanh

بنابراین استفاده از تابع tanh به‌عنوان تابع فعال‌ساز، امکان تصمیم‌گیری نرم و پیوسته را فراهم می‌آورد. با این حال، نحوه‌ی ترکیب خروجی نورون‌ها نقش حیاتی در صحت تصمیم‌گیری دارد. استفاده از min به‌عنوان ترکیب‌کننده‌ی خروجی‌ها عملکرد بسیار بهتری نسبت به حاصل‌ضرب دارد و منطبق با منطق AND در فضای پیوسته عمل می‌کند.

ب.۲) بررسی اثر تابع فعالساز RELU

در این مرحله، هدف بررسی تأثیر استفاده از تابع فعال‌ساز ReLU به‌همراه میانگین هندسی خروجی سه نورون تصمیم‌گیرنده برای شناسایی ناحیه‌ی مثلثی مشخص‌شده در فضای دوبعدی است.
اکنون به‌جای استفاده از نورون خروجی با تابع آستانه‌ی گسسته، از تابع ReLU استفاده می‌کنیم که تعریف آن به صورت زیر است و مقدار هر سه نورون مرزی ابتدا توسط ReLU نرمال می‌شوند.

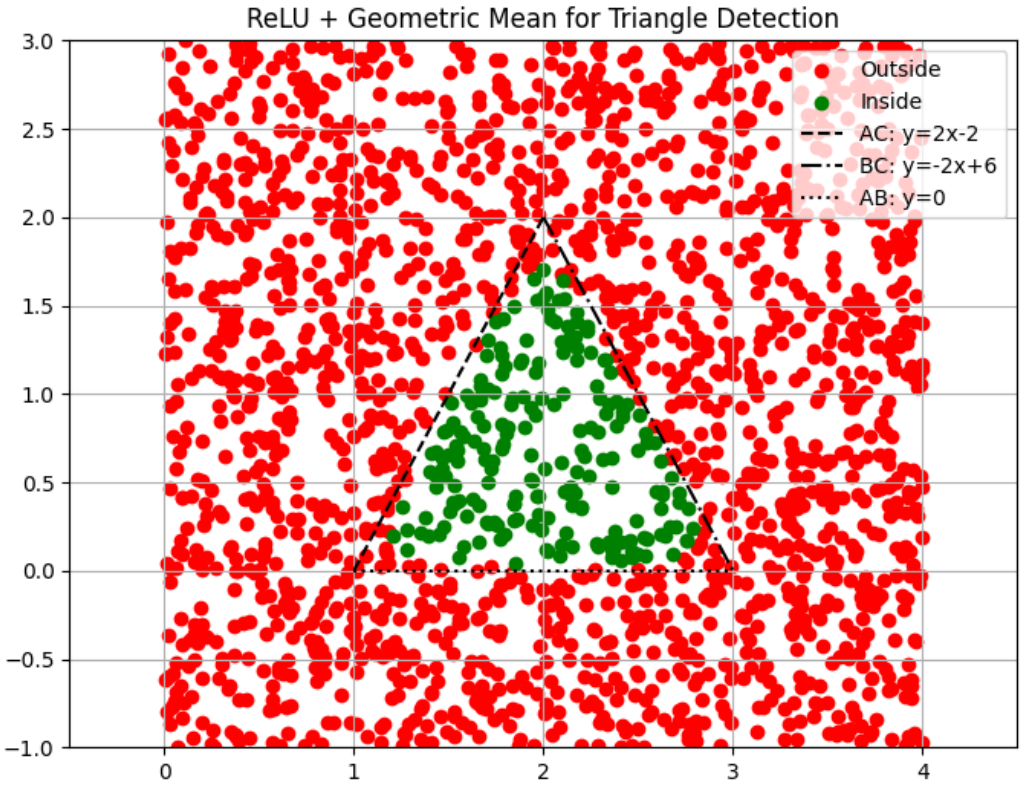
$$ReLU(x) = \begin{cases} x, & x \geq 0 \\ 0, & x < 0 \end{cases}$$

به‌جای انجام عملیات ضرب یا min که در روش‌های قبلی برای ترکیب سه خروجی استفاده می‌شد، این‌بار میانگین هندسی به کار گرفته شده است:

$$z = \sqrt[3]{z_1 * z_2 * z_3}$$

سپس این مقدار با یک آستانه‌ی ساده (مثلاً ۰.۵) مقایسه شده و تصمیم‌گیری نهایی صورت می‌گیرد.

نتیجه پیاده سازی کلاسبندی با این تابع فعالساز به صورت زیر می باشد:



شکل ۷_۱ نتایج کلاسبندی با منطق میانگین هندسی و تابع فعالساز ReLU

همانطورکه مشاهده شد، خروجی‌های حاصل از این روش، ناحیه مثلثی را به‌درستی شناسایی می‌کند، اما همچنان ناحیه‌ی تصمیم‌گیری دارای نویز و عدم قطعیت بود. به‌خصوص نقاط مرزی مثلث یا نواحی نزدیک به اضلاع به‌درستی طبقه‌بندی نشدند. علت این امر آن است که در تابعReLU ، مقادیر مثبت بدون محدودیت افزایش می‌یابند و میانگین هندسی ممکن است تحت تأثیر یک نورون با مقدار بزرگ قرار گیرد و تصمیم اشتباه بگیرد.

سوال ۲

بخش ۱.۱.۲

دیتاست «weather prediction» توسط نویسندگان مقاله برای آموزش عملی یادگیری ماشین و یادگیری عمیق طراحی شده و شامل داده‌های آب‌وهوایی از ۱۸ شهر اروپایی است. ساختار ساده، حجم مناسب و تنوع اهداف پیش‌بینی در این مجموعه باعث شده برای آموزش‌های hands-on در محیط‌های آکادمیک بسیار مناسب باشد. در این دیتاست برای هر شهر، ویژگی‌هایی مانند پوشش ابری، رطوبت، فشار، تابش خورشیدی، بارش، آفتاب، دمای میانگین، حداقل و حداکثر دما ارائه شده است. همچنین ستون‌های مشترک تاریخ و ماه برای تمام نمونه‌ها وجود دارد.

بخش ۲.۱.۲

اطلاعات شهر های TOURS, MONTELIMAR, PERPIGNAN از کشور فرانسه در این دیتاست موجود می باشد .

بخش ۳.۱.۲

دیتافریم نهایی df_france شامل ۳۶۵۴ نمونه یا ردیف داده مربوط به سه شهر فرانسوی مذکوراست. این داده‌ها بازه‌ای حدود ده سال را دربر می‌گیرند که با توجه به تعداد نمونه‌ها، ثبت روزانه داده‌ها از ابتدای سال ۲۰۰۰ تا پایان ۲۰۰۹ صورت گرفته است. هر شهر در این مجموعه دارای ۸ ویژگی آب‌وهوایی می‌باشد و تنها ویژگی cloud_cover برای هیچ‌یک از این سه شهر موجود نیست. به‌همین دلیل، به‌جای ۳ × ۹ = ۲۷ ستون مربوط به شهرها، تنها ۲۴ ستون برای ویژگی‌ها به همراه دو ستون تاریخ (DATE) و ماه (MONTH) در دیتافریم نهایی وجود دارد.

همچنین همانطور که در بخش ۲.۲.۴ این مقاله عنوان شده است، از روش max normalization _ min برای پیش پردازش دادگان استفاده شده است که بر روی داده این ۳ شهر نیز اعمال می کنیم. برای این منظور ابتدا داده های آموزش و آزمون را جدا کرده، minmax scaler را بر روی داده آموزش fit کرده و بر روی داده تست transform میکنیم.

همچنین لازم به ذکر است که از آخرین سطر موجود در این دیتاست که مربوط به سال ۲۰۱۰ می باشد استفاده ای نشده است.

بخش ۴.۱.۲

در این بخش به منظور تقسیم دادگان آموزش و تست به روش sliding window برای هر نمونه، پنج روز متوالی از داده‌ها به‌عنوان ورودی مدل در نظر گرفته شد و مقدار مربوط به روز بعدی به‌عنوان خروجی هدف تعریف گردید. این کار باعث شد مجموعه داده‌ها به‌صورت مجموعه‌ای از دنباله‌های زمانی شکل بگیرد که هر کدام شامل پنج روز پشت‌سرهم از اطلاعات سه شهر هستند. در نتیجه، برای داده‌های آزمون که مربوط به سال ۲۰۰۹ و شامل ۳۶۵ روز بود، با توجه به نیاز پنجره لغزان به روزهای قبلی، تنها از ۳۶۱ روز آن می‌توان برای ایجاد نمونه‌های قابل استفاده در مدل بهره گرفت، ابعاد دیتافریم های آزمون و آموزش به صورت زیر می باشد.

```
X_train shape = (3284, 5, 24)
y_train shape = (3284, 24)
X_test shape  = (361, 5, 24)
y_test shape  = (361, 24)
```

شکل ۸_۲ ابعاد دیتافریم های آموزش و آزمون

بخش ۲.۲

مطابق آنچه خواسته شده اطلاعات تنها شهر Perpignan انتخاب میکنیم و ابعاد دیتافریم های آموزش و تست به صورت زیر خواهد شد (کاهش تعداد فیچر ها از ۲۴ به ۸)

```
X_train shape = (3284, 5, 8)
y_train shape = (3284, 8)
X_test shape  = (361, 5, 8)
y_test shape  = (361, 8)
```

شکل ۹_۲ ابعاد دیتافریم های آموزش و آزمون برای شهر Perpignan

بخش ۱.۲.۲

یادگیری ماشین مشارکتی (Collaborative Machine Learning) یک رویکرد پیشرفته در مدل‌سازی است که در آن برای پیش‌بینی یک متغیر هدف در یک موقعیت مکانی خاص، از داده‌های جمع‌آوری‌شده از چندین موقعیت مکانی دیگر نیز استفاده می‌شود. این روش بر این اصل استوار است که همبستگی‌های فضایی و زمانی میان مناطق مختلف می‌تواند اطلاعات ارزشمندی را در اختیار مدل قرار دهد که صرفاً با استفاده از داده‌های تاریخی یک منطقه قابل دستیابی نیست. هدف اصلی این رویکرد، افزایش دقت و پایداری پیش‌بینی‌ها از طریق غنی‌سازی مجموعه داده‌های ورودی و بهره‌گیری از الگوهای مکانی پنهان است.

در مقاله، این مفهوم در قالب یک سیستم پیش‌بینی مشارکتی آب و هوا پیاده‌سازی شده است. برای این منظور، داده‌های آب و هوایی از چندین منطقه در کشور موریس جمع‌آوری شده و مدل‌های یادگیری ماشین به گونه‌ای اصلاح شده‌اند که بتوانند ورودی‌ها را از مناطق مختلف به صورت هم‌زمان پردازش کنند. به طور مشخص، در آزمایش‌های انجام‌شده، برای پیش‌بینی وضعیت جوی یک منطقه هدف، داده‌های دو منطقه مجاور دیگر نیز در فرآیند آموزش مدل لحاظ گردید. نتایج به دست آمده نشان داد که مدل‌های مشارکتی توانستند به طور میانگین خطای پیش‌بینی را تا ۵ درصد کاهش دهند که این امر، کارایی رویکرد مشارکتی در افزایش دقت سیستم‌های پیش‌بینی آب و هوا را تایید می‌کند .

بخش ۳.۲

در این بخش با استفاده از کتابخانهPyTorch ، پیاده‌سازی مدل عصبی را آغاز می کنیم. با توجه به اینکه تصمیم گرفتیم از یک مدل با ۸ خروجی برای پیش‌بینی هم‌زمان تمام ویژگی‌های شهر Perpignan استفاده کنیم، ابتدا داده‌های آماده‌شده با sliding window به آرایه‌های PyTorch تبدیل شدند. سپس، برای سازگاری با شبکه‌ی عصبی خطی، ساختار ورودی‌ها از حالت سه‌بعدی (توالی زمانی) به حالت دوبعدی (ترتیبی) بازسازی شد. در ادامه، از کلاس DataLoader برای ایجاد دسته‌های آموزشی و آزمون استفاده شد تا فرایند آموزش در دسته‌های کوچک و قابل کنترل انجام گیرد. در نهایت، ساختار شبکه‌ی عصبی با استفاده از یک کلاس سفارشی به نام WeatherNet تعریف شد که شامل یک لایه پنهان با تابع فعال‌ساز ReLU و یک لایه خروجی خطی برای تولید ۸ مقدار خروجی پیش‌بینی‌شده به‌طور هم‌زمان بود.

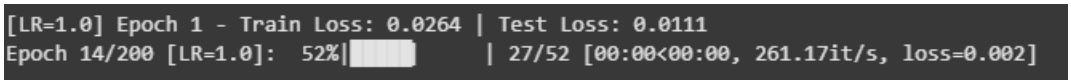
سپس، در مرحله بعد، یک تابع آموزش عمومی طراحی شد که امکان آموزش مدل را با نرخ‌های یادگیری مختلف فراهم می‌کرد. در این تابع، ابتدا شبکه‌ی عصبی با ابعاد ورودی برابر با تعداد کل ویژگی‌های پنجره زمانی و ابعاد خروجی برابر با ۸ مقدار هدف تعریف شد. برای بهینه‌سازی مدل از الگوریتم SGD استفاده شد و تابع خطا نیز

MSE (میانگین مربعات خطا) در نظر گرفته شد. در هر epoch ، ابتدا مدل در حالت آموزش قرار گرفت و با پیمایش داده‌ها به‌صورت batch مقادیر پیش‌بینی‌شده با مقادیر واقعی مقایسه و وزن‌ها با استفاده از backpropagation به‌روزرسانی شدند. سپس مدل در حالت ارزیابی قرار گرفت و خطای آزمون روی کل داده‌ی تست بدون اعمال گرادیان محاسبه شد. در هر epoch ، خطای آموزش و آزمون ثبت و در بازه‌های مشخص گزارش شد تا بتوان روند یادگیری مدل را به‌صورت کمی پایش کرد.

در نهایت، در مرحله سوم، فرآیند آموزش مدل برای سه مقدار متفاوت نرخ یادگیری شامل 1 و 10^{-3} و 10^{-8} اجرا شد تا تأثیر این پارامتر کلیدی بر روند یادگیری مدل بررسی شود. برای هر مقدار نرخ یادگیری، مدل به‌صورت مجزا آموزش داده شد و نمودارهای مربوط به تغییرات خطای آموزش و آزمون در طول ۲۰۰ دوره رسم گردیدند. این نمودارها امکان مقایسه‌ی مستقیم کارایی مدل در شرایط مختلف را فراهم کرده و به تحلیل بهتر رفتار مدل در مواجهه با مقادیر مختلف گام به‌روزرسانی کمک می‌کنند. همچنین، مقادیر نهایی خطای آموزش و آزمون پس از پایان هر آموزش نمایش داده شد تا بتوان بهترین نرخ یادگیری را با توجه به دقت نهایی و روند همگرایی انتخاب کرد.

بخش ۴.۲

مطابق آنچه در بخش ۳.۲ توضیح داده شد، نتایج کد پیاده سازی شده و خواسته های مسئله در ادامه آورده خواهد شد برای نمایش progress bar حین آموزش با استفاده از کتابخانه tqdm استفاده شد که در شکل زیر مشاهده می شود.



شکل ۱۰_۲ نمایش progress bar حین آموزش

همانطور که در تصویر بالا نیز مشاهده میشود مقادیر خطای آموزش و آزمون حین آموزش پرینت می شوند که به ازای ۳ مقدار نرخ یادگیری نتایج به صورت تصاویر زیر می باشد.

```
[LR=1.0] Epoch 1 - Train Loss: 0.0264 | Test Loss: 0.0111
[LR=1.0] Epoch 20 - Train Loss: 0.0015 | Test Loss: 0.0019
[LR=1.0] Epoch 40 - Train Loss: 0.0011 | Test Loss: 0.0010
[LR=1.0] Epoch 60 - Train Loss: 0.0009 | Test Loss: 0.0014
[LR=1.0] Epoch 80 - Train Loss: 0.0005 | Test Loss: 0.0004
[LR=1.0] Epoch 100 - Train Loss: 0.0006 | Test Loss: 0.0003
[LR=1.0] Epoch 120 - Train Loss: 0.0006 | Test Loss: 0.0012
[LR=1.0] Epoch 140 - Train Loss: 0.0005 | Test Loss: 0.0003
[LR=1.0] Epoch 160 - Train Loss: 0.0003 | Test Loss: 0.0004
[LR=1.0] Epoch 180 - Train Loss: 0.0004 | Test Loss: 0.0005
[LR=1.0] Epoch 200 - Train Loss: 0.0001 | Test Loss: 0.0003
```

شکل ۱۱_۲ خطای آموزش و آزمون حین آموزش شبکه با نرخ یادگیری ۱

```
[LR=0.001] Epoch 1 - Train Loss: 0.2363 | Test Loss: 0.2213
[LR=0.001] Epoch 20 - Train Loss: 0.0387 | Test Loss: 0.0374
[LR=0.001] Epoch 40 - Train Loss: 0.0268 | Test Loss: 0.0262
[LR=0.001] Epoch 60 - Train Loss: 0.0251 | Test Loss: 0.0246
[LR=0.001] Epoch 80 - Train Loss: 0.0238 | Test Loss: 0.0233
[LR=0.001] Epoch 100 - Train Loss: 0.0226 | Test Loss: 0.0222
[LR=0.001] Epoch 120 - Train Loss: 0.0215 | Test Loss: 0.0212
[LR=0.001] Epoch 140 - Train Loss: 0.0205 | Test Loss: 0.0202
[LR=0.001] Epoch 160 - Train Loss: 0.0195 | Test Loss: 0.0194
[LR=0.001] Epoch 180 - Train Loss: 0.0187 | Test Loss: 0.0186
[LR=0.001] Epoch 200 - Train Loss: 0.0179 | Test Loss: 0.0178
```

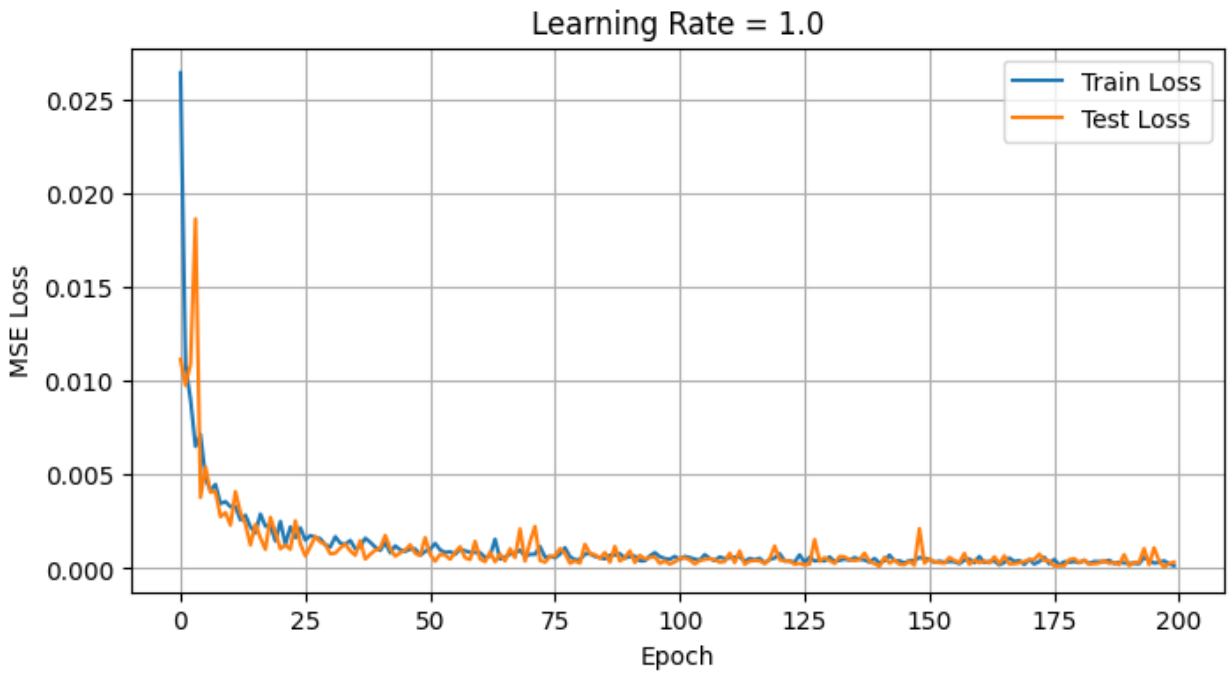
شکل ۱۲_۲ خطای آموزش و آزمون حین آموزش شبکه با نرخ یادگیری ۰.۰۰۱

```
[LR=1e-08] Epoch 1 - Train Loss: 0.3212 | Test Loss: 0.3162
[LR=1e-08] Epoch 20 - Train Loss: 0.3212 | Test Loss: 0.3162
[LR=1e-08] Epoch 40 - Train Loss: 0.3213 | Test Loss: 0.3162
[LR=1e-08] Epoch 60 - Train Loss: 0.3215 | Test Loss: 0.3162
[LR=1e-08] Epoch 80 - Train Loss: 0.3218 | Test Loss: 0.3162
[LR=1e-08] Epoch 100 - Train Loss: 0.3211 | Test Loss: 0.3162
[LR=1e-08] Epoch 120 - Train Loss: 0.3213 | Test Loss: 0.3162
[LR=1e-08] Epoch 140 - Train Loss: 0.3215 | Test Loss: 0.3162
[LR=1e-08] Epoch 160 - Train Loss: 0.3214 | Test Loss: 0.3162
[LR=1e-08] Epoch 180 - Train Loss: 0.3215 | Test Loss: 0.3162
[LR=1e-08] Epoch 200 - Train Loss: 0.3211 | Test Loss: 0.3162
```

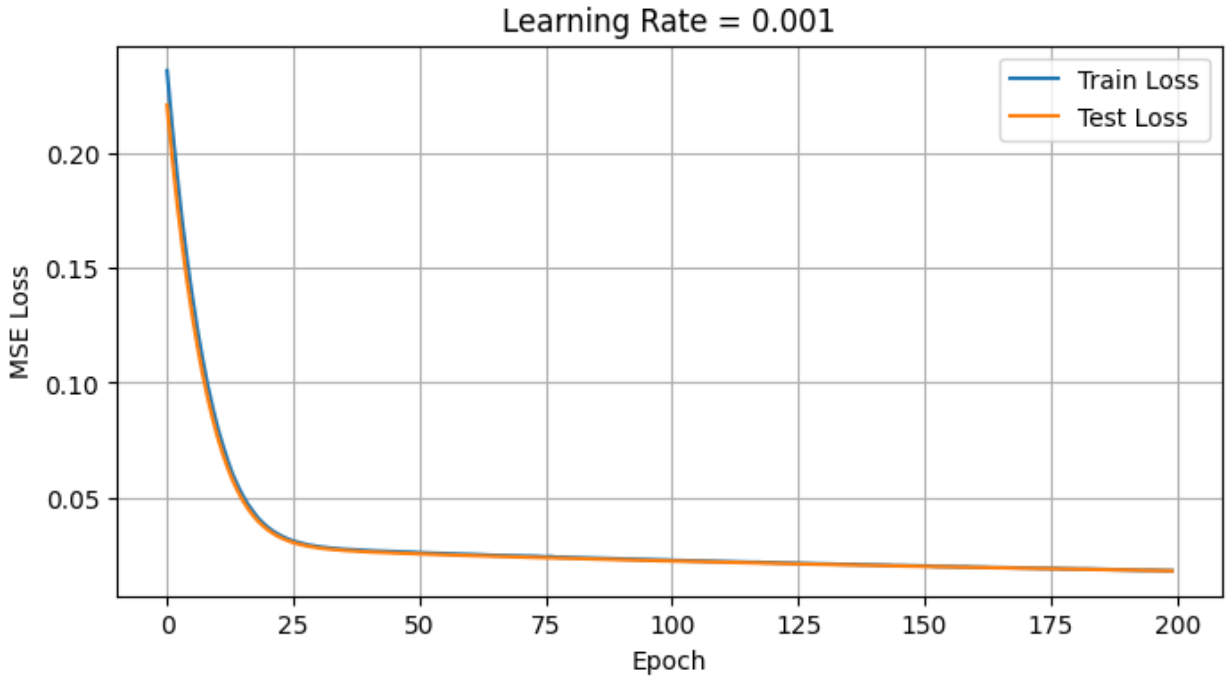
شکل ۱۳_۲ خطای آموزش و آزمون حین آموزش شبکه با نرخ یادگیری 10e-8

بخش ۱.۴.۲

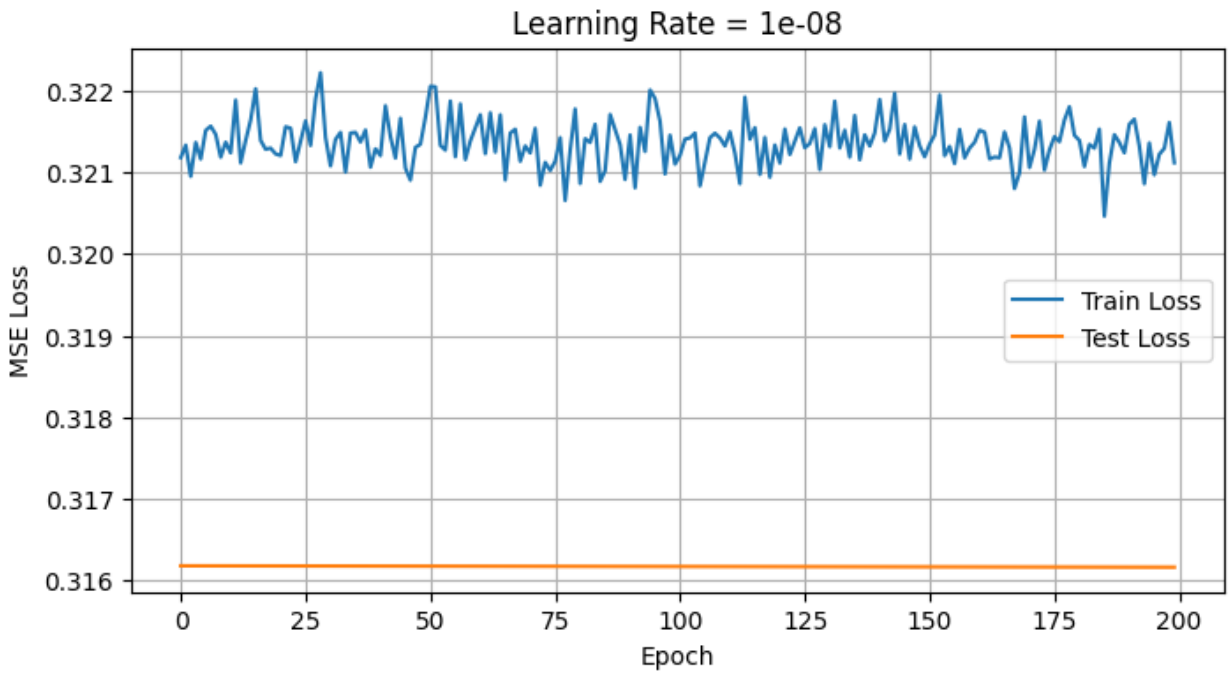
در بخش امتیازی نیز نمودار خطا آزمون و آموزش به ازای ۳ مقدار نرخ یادگیری به صورت تصاویر زیر می باشد:



شکل ۱۴_۲ نمودار خطا آموزش و آزمون به ازای نرخ یادگیری ۱



شکل ۱۵_۲ نمودار خطا آموزش و آزمون به ازای نرخ یادگیری ۰.۰۰۱



شکل ۱۶_۲ نمودار خطا آموزش و آزمون به ازای نرخ یادگیری 10e-8

بخش ۲.۴.۲

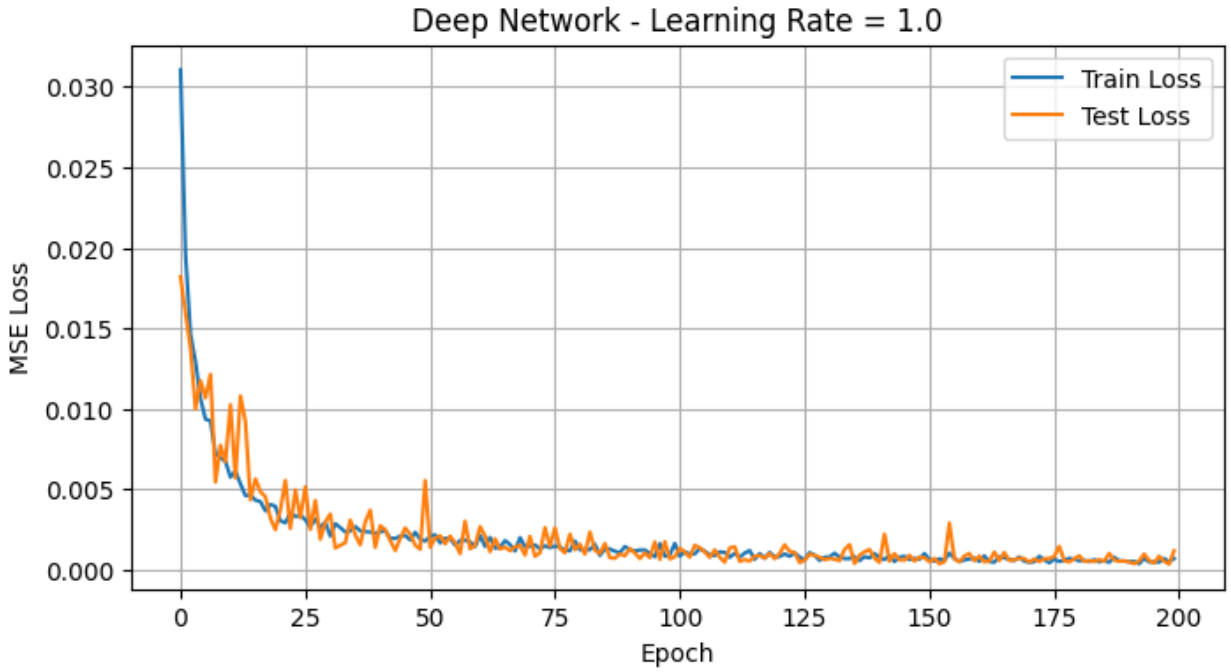
همانطور که مشاهده می شود، در نرخ یادگیری ۱ مدل به‌سرعت به حداقل خطا نزدیک شده و نوسانات جزئی در خطای آزمون دارد، که بیانگر سرعت بالا در یادگیری است ولی ممکن است کمی ناپایداری ایجاد کند. در نرخ یادگیری ۰.۰۰۱ مدل به‌صورت یکنواخت و کنترل‌شده همگرایی داشته و بدون نوسان به سمت کاهش خطا حرکت کرده است؛ این نمودار بهترین تعادل بین دقت و پایداری را نشان می‌دهد. در مقابل، نرخ یادگیری بسیار پایین 10e-8 باعث شده مدل تقریباً هیچ بهبودی در طول آموزش نداشته باشد؛ خطای آموزش بالا مانده و تقریباً ثابت است، که نشان‌دهنده‌ی حرکت بسیار کند یا توقف در یادگیری است. بنابراین، نرخ یادگیری ۰.۰۰۱ مناسب‌ترین مقدار برای ادامه‌ی آموزش و استفاده در مدل نهایی است.

بخش ۵.۲

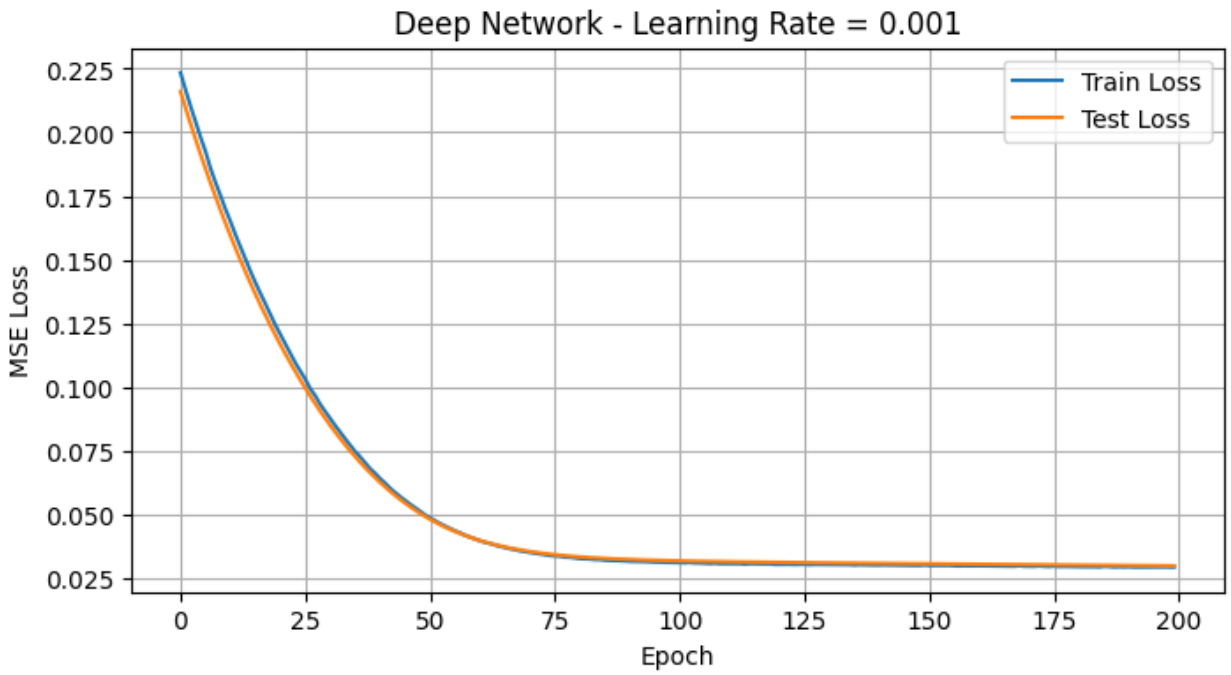
در این مرحله، با هدف بررسی تأثیر افزایش عمق شبکه‌ی عصبی بر عملکرد مدل، ساختار جدیدی با سه لایه‌ی پنهان طراحی می کنیم. مدل جدید شامل یک معماری متوالی با لایه‌های خطی و تابع فعال‌ساز ReLU به صورت زیر بود:

لایه اول با ۱۲۸ نورون، لایه دوم با ۶۴ نورون، لایه سوم با ۳۲ نورون و در نهایت یک لایه خروجی با ۸ نورون و تابع فعال‌ساز خطی برای پیش‌بینی هم‌زمان ۸ ویژگی هواشناسی شهر Perpignan طراحی شد. داده‌های ورودی، مانند مرحله‌ی قبل، شامل پنجره‌هایی از طول ۵ روز بر روی داده‌های نرمال‌سازی‌شده بودند.

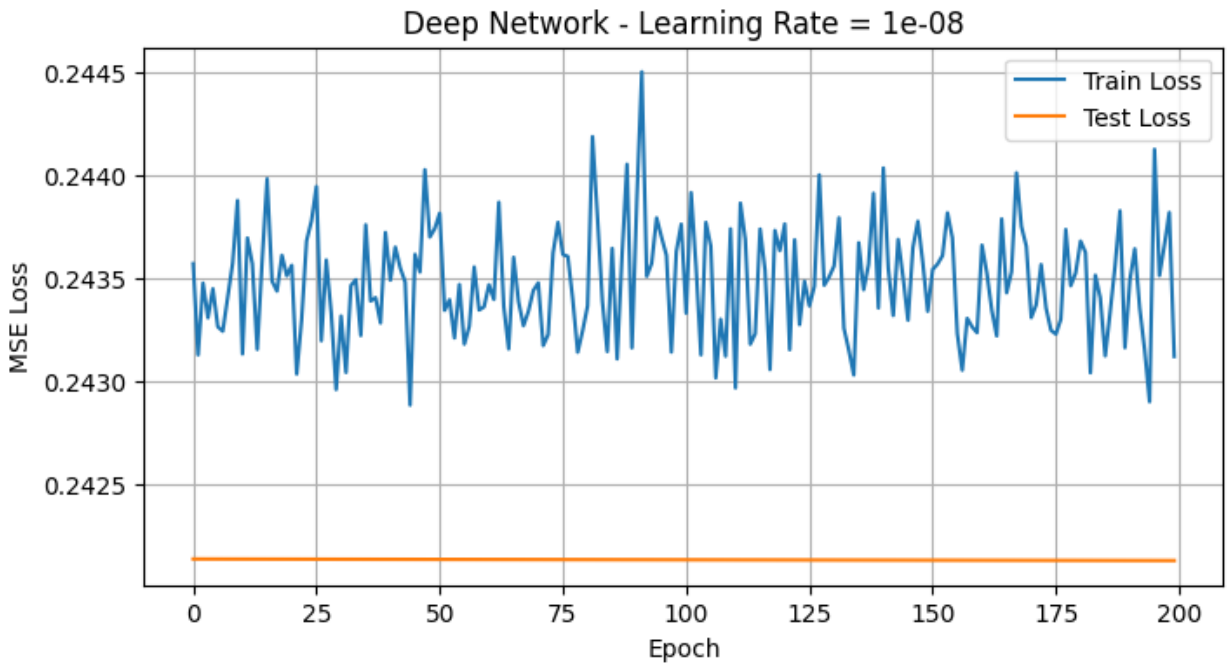
آموزش مدل برای سه مقدار نرخ یادگیری مختلف 1 و 10^{-3} و 10^{-8} انجام گرفت. نمودارهای زیر روند تغییرات خطای آموزش و آزمون را در طول ۲۰۰ دوره‌ی یادگیری برای هر مقدار نرخ یادگیری نمایش می‌دهند.



شکل ۱۷_۲ نمودار خطا آموزش و آزمون شبکه عمیق به ازای نرخ یادگیری ۱



شکل ۱۸_۲ نمودار خطا آموزش و آزمون شبکه عمیق به ازای نرخ یادگیری ۰.۰۰۱



شکل ۱۹_۲ نمودار خطا آموزش و آزمون شبکه عمیق به ازای نرخ یادگیری 10e-8

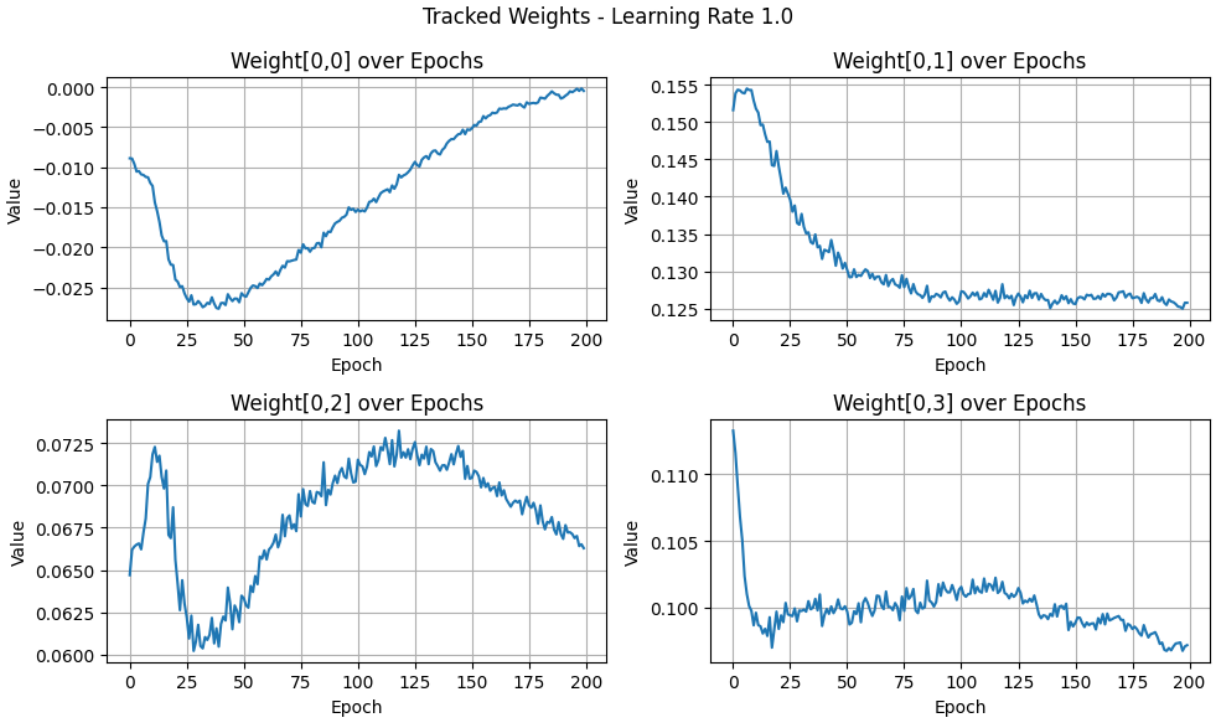
با مقایسه‌ی نتایج این مرحله با مدل قبلی، مشخص شد که افزایش عمق شبکه در این مسئله تفاوت محسوسی در کاهش خطا یا بهبود دقت مدل ایجاد نکرد. دلیل اصلی این موضوع را می‌توان در سادگی نسبی داده‌ها، حجم محدود ورودی‌ها و پیچیدگی پایین روابط میان ویژگی‌ها دانست. به عبارت دیگر، ظرفیت مدل ساده نیز برای یادگیری این مسئله کافی بوده و مدل عمیق‌تر به‌دلیل نداشتن فضای یادگیری پیچیده‌تر، بهبود خاصی در عملکرد نشان نداد. بنابراین، در چنین مسائل نسبتاً ساده، استفاده از مدل‌های سبک‌تر می‌تواند کارایی و پایداری بهتری داشته باشد.

بخش ۶.۲

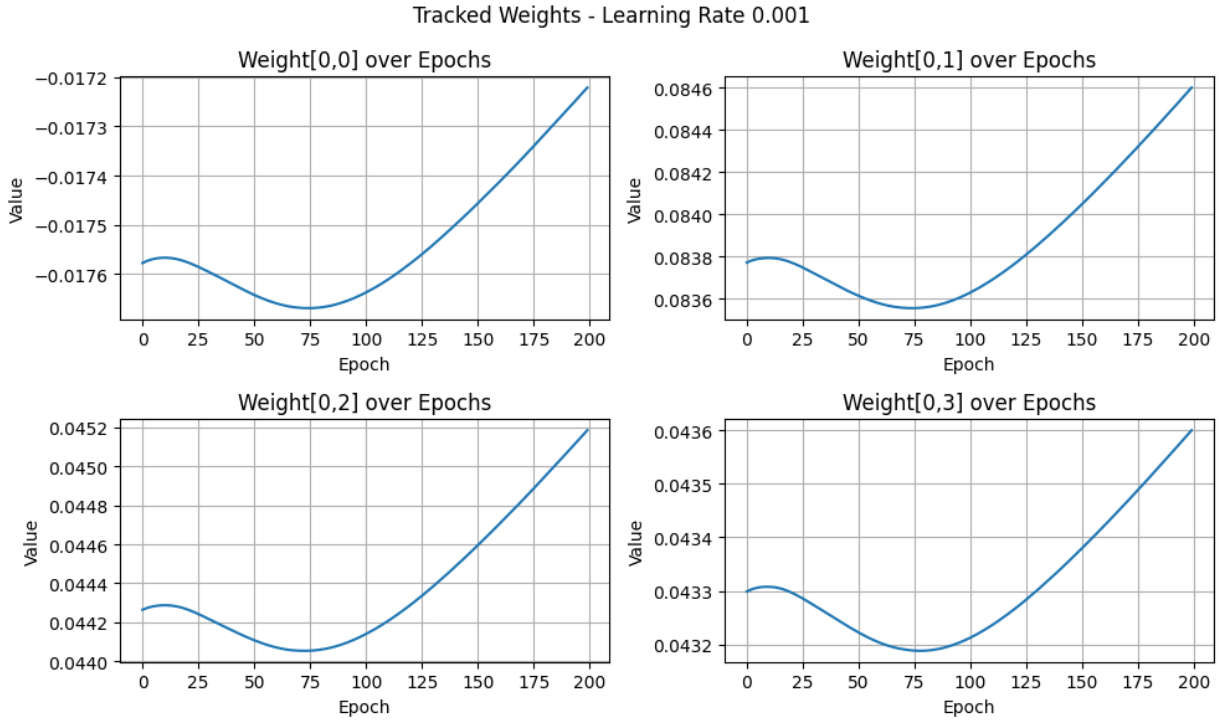
در این بخش به‌منظور تحلیل رفتار مدل در حین یادگیری، تعدادی از وزن‌های شبکه عصبی ذخیره و تغییرات آن‌ها در طول فرایند آموزش ثبت شد. مدل مورد استفاده، همان مدل عمیق تعریف‌شده در بخش ۵.۲ بوده و تحلیل بر اساس همان ساختار انجام شده است.

برای این منظور، چهار مقدار از وزن‌های لایه‌ی اول شبکه (Linear(40, 128)) انتخاب شدند؛ این وزن‌ها به‌صورت w[0,0], w[0,1], w[0,2], w[0,3] مشخص شده‌اند. در اینجا، w[0,0] به معنای وزن اتصال بین اولین ویژگی ورودی و اولین نورون در لایه‌ی بعدی است. سایر وزن‌ها نیز به‌صورت مشابه، بیانگر اتصالات اولیه میان ورودی‌ها و نورون‌های لایه‌ی اول پنهان هستند.

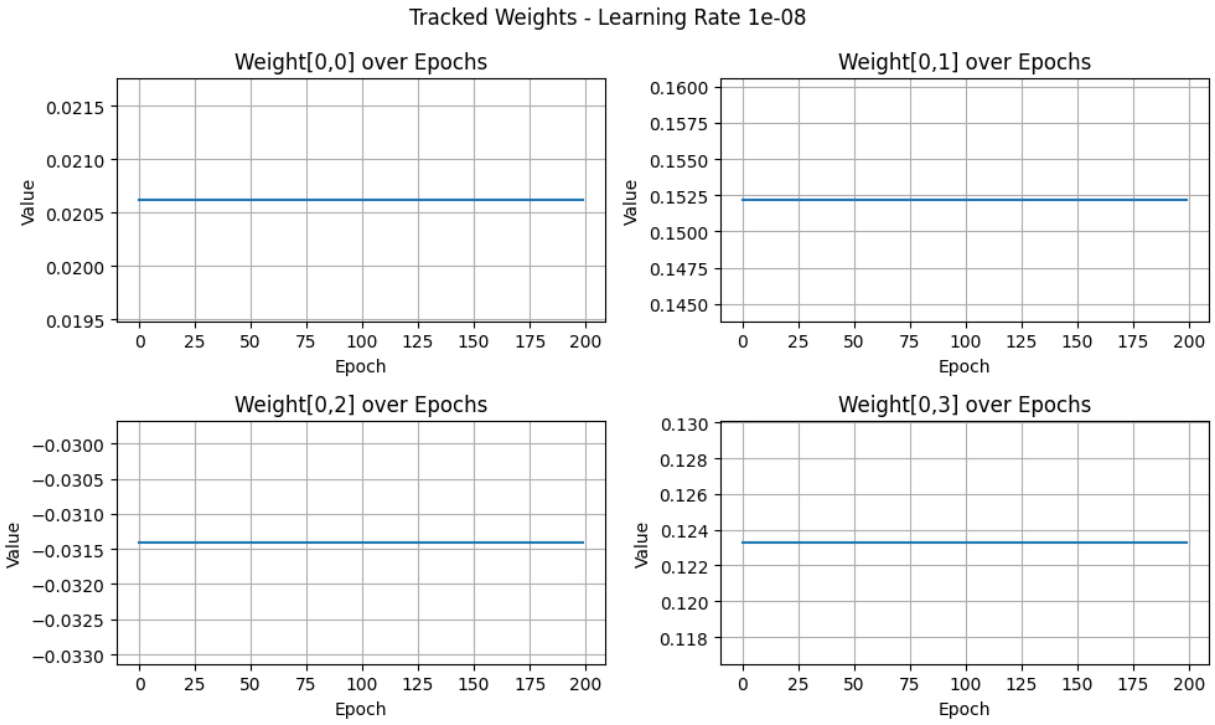
نتایج ردیابی تغییر وزن ها به صورت زیر می باشد.



شکل ۲۰_ ۲ نمودار تغییر وزن ها به ازای نرخ یادگیری ۱



شکل ۲۱_ ۲ نمودار تغییر وزن ها به ازای نرخ یادگیری ۰.۰۰۱



شکل ۲۲_ ۲ نمودار تغییر وزن ها به ازای نرخ یادگیری 10e-8

همانطور که مشاهده می شود، تغییرات چهار وزن ابتدایی لایه اول شبکه عصبی در طول آموزش با سه نرخ یادگیری مختلف بررسی شد. در نرخ یادگیری ۱، وزن‌ها تغییرات شدید و نوسانات زیادی داشتند که نشان‌دهنده سرعت بالای یادگیری همراه با بی‌ثباتی است. در مقابل، با نرخ یادگیری ۰.۰۰۱ روند تغییر وزن‌ها بسیار نرم، پیوسته و همگرا بود که نشان‌دهنده یادگیری پایدار و مؤثر شبکه است. اما در نرخ بسیار پایین 10e-8، وزن‌ها تقریباً بدون تغییر باقی ماندند که نشان می‌دهد مدل عملاً هیچ‌گونه یادگیری مؤثری نداشته است.

در ادامه به توضیح تاثیر استفاده از مفاهیم خواسته شده پرداخته خواهد شد.

۱- پیش‌پردازش داده‌ها:(Data Preprocessing)

پیش‌پردازش داده‌ها یکی از ابتدایی‌ترین و در عین حال حیاتی‌ترین مراحل در طراحی مدل‌های یادگیری ماشین است که هدف آن تبدیل داده‌های خام به شکلی قابل پردازش و مؤثر برای یادگیری است. این فرایند شامل حذف داده‌های پرت، نرمال‌سازی، نرمال‌سازی ویژگی‌ها، تکمیل مقادیر گمشده، یا کدگذاری متغیرهای طبقه‌ای می‌شود. در مدل‌های شبکه عصبی، نرمال‌سازی داده‌ها باعث می‌شود که گرادیان‌ها در حین آموزش پایدارتر باشند و سرعت همگرایی افزایش یابد. در پروژه حاضر، داده‌ها با استفاده از روش Min-Max Normalization به بازه [۰,۱] مقیاس‌دهی شدند تا اختلاف مقیاس بین ویژگی‌ها تأثیری منفی بر یادگیری شبکه نگذارد.

۲- مقداردهی اولیه وزن‌ها:(Weight Initialization)

مقداردهی اولیه وزن‌ها نقش مهمی در عملکرد اولیه و مسیر همگرایی مدل دارد. انتخاب تصادفی وزن‌ها بدون اصول مشخص می‌تواند باعث توقف آموزش یا یادگیری کند شود. روش‌های استانداردی مانند Xavier Initialization(برای فعال‌سازهای تانژانت هایپربولیک) و) He Initialization برای (ReLU پیشنهاد می‌شوند که براساس تعداد ورودی و خروجی هر نورون طراحی شده‌اند. این روش‌ها در دوره یادگیری عمیق دانشگاه استنفورد به‌عنوان پایه‌ای‌ترین اصول مطرح شده‌اند. در پروژه ما، مقداردهی اولیه به صورت پیش‌فرض PyTorch (که خود از توزیع‌های نرمال یا یکنواخت با مقیاس متناسب استفاده می‌کند) انجام شد، اما می‌توان در نسخه‌های آینده با تعیین صریح این الگوریتم‌ها، کنترل بیشتری بر روند آموزش داشت.

۳- نرمال‌سازی دسته‌ای:(Batch Normalization)

Batch Normalizationیک تکنیک تثبیت‌کننده در طول آموزش شبکه‌های عمیق است که با نرمال‌سازی خروجی هر لایه برای هر batch ، باعث کاهش وابستگی به مقداردهی اولیه وزن‌ها و شتاب‌گیری یادگیری می‌شود. این روش همچنین به مدل کمک می‌کند تا در برابر تغییرات در توزیع داده‌های میانی (internal covariate shift) مقاوم‌تر باشد. علاوه بر آن، گاهی نقش regularizer نیز ایفا می‌کند. در پروژه فعلی، از این تکنیک در ساختار مدل استفاده نشده است. با توجه به عمق نسبی شبکه ما، اضافه کردن BatchNorm می‌تواند در بهبود همگرایی سریع‌تر و پایداری در آموزش مؤثر واقع شود.

۴- منظم‌سازی:(Regularization)

Regularizationمجموعه‌ای از تکنیک‌ها برای کنترل پیچیدگی مدل و جلوگیری از بیش‌برازش (Overfitting) است. این روش‌ها مانند L1 یا L2 regularization با اضافه کردن جریمه به تابع هزینه، یا روش‌هایی مانند Early Stopping، مانع از تطابق بیش‌ازحد مدل با داده‌های آموزشی می‌شوند. هدف نهاییregularization ، افزایش توان تعمیم مدل به داده‌های دیده‌نشده است. در این پروژه، به‌صورت مستقیم از تکنیک‌های منظم‌سازی بهره گرفته نشده، زیرا داده‌ها ساختار ساده‌ای دارند و در مراحل تحلیل، نشانه‌ای از بیش‌برازش جدی مشاهده نشد. با این حال، استفاده از L2 یا Early Stopping در شبکه‌های پیچیده‌تر آینده توصیه می‌شود.

۵-دراپ‌اوت:(Dropout)

Dropoutیک تکنیک قدرتمند و ساده برای جلوگیری از بیش‌برازش در شبکه‌های عصبی است. در این روش، در هر مرحله‌ی آموزش، تعدادی از نورون‌ها به‌صورت تصادفی غیرفعال می‌شوند. این کار باعث می‌شود که شبکه به وابستگی شدید به نورون‌های خاص گرفتار نشود و مدل نهایی، ترکیب متنوعی از مسیرهای مختلف یادگیری را یاد بگیرد Dropout .نوعی regularization محسوب می‌شود و در معماری‌های پیچیده با داده‌های زیاد و نویز بالا بسیار مؤثر است. در پروژه‌ی حاضر، از Dropout استفاده نشده است، زیرا هدف اصلی بررسی معماری پایه و تحلیل نرخ‌های یادگیری بود، اما افزودن آن در مدل‌های آینده، به‌ویژه در صورت وجود داده‌های متنوع‌تر، می‌تواند بسیار سودمند باشد.

سوال ۳

تعریف محیط

درابتدا، محیطی به نام SimpleWumpusWorld طراحی و پیاده‌سازی می کنیم. این محیط در قالب یک شبکه‌ی ۴×۴ تعریف شده است که شامل عناصر مختلفی نظیر طلا(Gold) ، چاله‌ها(Pits) ، موجود خطرناک (Wumpus) و عامل (Agent) است. تمامی موقعیت‌های این عناصر در ابتدای هر اپیزود به‌صورت ثابت و از پیش‌تعیین‌شده مقداردهی می‌شوند تا فرآیند یادگیری در یک محیط پایدار انجام گیرد.

در این محیط، موقعیت شروع عامل برابر با (3, 0) است. هدف عامل، رسیدن به سلول حاوی طلا در مختصات (1, 2) است. سه چاله در سلول‌های (2, 2) ,(3, 0)و (0, 0)قرار گرفته‌اند که در صورت ورود عامل به هر یک از آن‌ها، اپیزود با شکست و دریافت پاداش 1000-خاتمه می‌یابد. همچنین یک Wumpus در موقعیت (1, 0) قرار دارد که مشابه چاله‌ها، مرگ‌آور محسوب می‌شود. در مقابل، رسیدن به طلا با پاداش 100+ همراه است. سایر سلول‌ها دارای پاداش پیش‌فرض 1- هستند تا عامل برای هر حرکت یک جریمه کوچک دریافت کرده و از پرسه‌زدن بی‌هدف اجتناب نماید.

تابع ()reset موقعیت اولیه عامل و وضعیت محیط را مقداردهی می‌کند و تابع step(action) با دریافت یک اقدام (بالا، پایین، چپ یا راست)، موقعیت عامل را به‌روزرسانی کرده و پاداش متناظر را بازمی‌گرداند. همچنین در صورت ورود به یک سلول نهایی (طلا یا تهدید)، وضعیت محیط به حالت پایان‌یافته (done = True) تغییر می‌یابد. تابع ()render نیز امکان نمایش وضعیت فعلی محیط را به‌صورت متنی و نمادین فراهم می‌سازد.

این طراحی ساده و ساختارمند باعث می‌شود عامل بتواند در یک محیط پایدار، سیاست بهینه خود را برای رسیدن به طلا و اجتناب از خطرات یاد بگیرد.

آ)

در ادامه، الگوریتم یادگیری تقویتی Q-learning برای آموزش عامل در محیط SimpleWumpusWorld پیاده‌سازی و اجرا شد. هدف اصلی عامل، یادگیری سیاستی بهینه برای حرکت در شبکه‌ی ۴×۴ به‌گونه‌ای است که از چاله‌ها و Wumpus اجتناب کرده و به طلا برسد.

یک Q-table سه‌بعدی با ابعاد (4, 4, 4) تعریف شد که هر عنصر آن، پاداش مورد انتظار از انتخاب یک اقدام خاص در یک حالت معین را نشان می‌دهد. عامل در هر گام با استفاده از استراتژی ε-greedy ، بین اکتشاف

(exploration) و بهره‌برداری (exploitation) تصمیم می‌گیرد. نرخ اکتشاف ε به‌صورت خطی از مقدار اولیه 1.0 تا مقدار حداقلی 0.01 کاهش یافت.

پارامترهای آموزش:Q-learning :

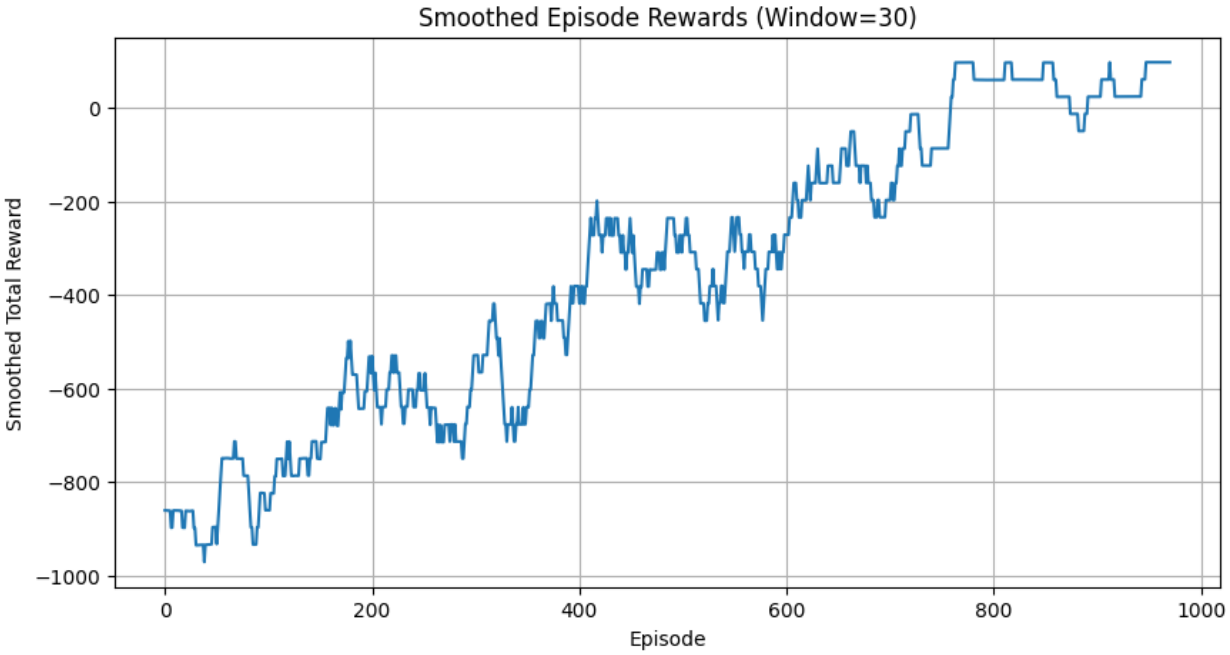
- تعداد اپیزودها : 1000
- حداکثر گام مجاز در هر اپیزود : 50
- نرخ یادگیری α: 0.1
- ضریب تخفیف آینده γ: 0.9
- نرخ اکتشاف اولیه ε: 1.0
- نرخ اکتشاف نهاییε_min: 0.01
- نحوه کاهشε : خطی، به‌صورت کاهش یکنواخت از ۱.۰ تا ۰.۰۱ در طول اپیزودها
- تعداد اعمال ممکن : 4 حرکت (بالا، پایین، چپ، راست)

الگوریتم Q-learning به‌منظور آموزش یک عامل (Agent) در محیطی ساده‌شده از دنیای Wumpus پیاده‌سازی شده است. محیط شامل یک شبکه ۴ در ۴ است که در آن عامل از موقعیت ابتدایی (۳, ۰) حرکت خود را آغاز می‌کند و هدف آن رسیدن به طلا در موقعیت (۱, ۲) است. در این مسیر، عامل باید از برخورد با سه چاله در موقعیت‌های (۰, ۳)، (۲, ۲) و (۰, ۰) و همچنین موجودی به‌نام ومپوس در خانه‌ی (۱, ۰) اجتناب کند. خانه‌ی طلا دارای پاداش مثبت ۱۰۰، چاله‌ها و ومپوس دارای جریمه سنگین ۱۰۰۰- و سایر خانه‌ها دارای پاداش پایه ۱- برای هر حرکت هستند.

به‌منظور یادگیری سیاست بهینه، یک جدول Q با ابعاد (۴×۴×۴) تعریف شده است که در آن، برای هر حالت (موقعیت عامل در محیط) و هر عمل ممکن (بالا، پایین، چپ، راست)، مقدار Q به‌روزرسانی می‌شود. عامل با استفاده از سیاست ε-greedy بین اکتشاف و بهره‌برداری تصمیم‌گیری می‌کند، به‌طوری‌که مقدار ε در طول ۱۰۰۰ اپیزود به‌صورت خطی از ۱.۰ به ۰.۰۱ کاهش می‌یابد. در هر اپیزود، عامل حداکثر ۵۰ حرکت فرصت دارد تا به هدف برسد یا با تهدیدی مواجه شود. الگوریتم آپدیت شدن Q_table به صورت فرمول زیر می باشد.

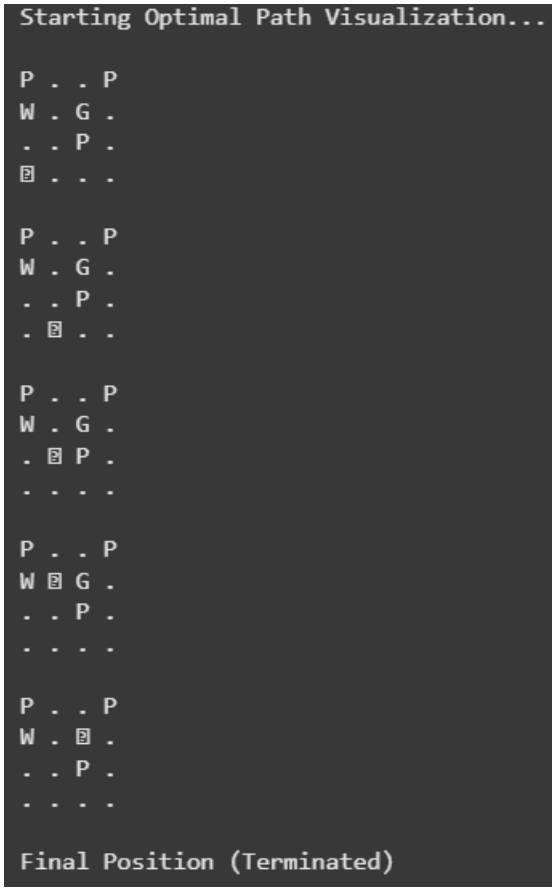
$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma \max_{a'} Q(s',a') - Q(s,a) \right]$$

نتایج حاصل از آموزش عامل با Q-learning در قالب نمودار پاداش اپیزودها نشان داد که عامل در ابتدا اغلب با چاله‌ها یا ومپوس برخورد کرده و پاداش‌های منفی دریافت می‌کرده است، اما به مرور زمان و با کاهش نرخ اکتشاف، موفق به یافتن مسیرهای امن‌تر و رسیدن به هدف شده است. مسیرهای بهینه توسط Q-table استخراج شدند و به‌صورت بصری نیز قابل مشاهده بودند. روند صعودی و پایدار پاداش‌ها در نمودار میانگین متحرک، بیانگر موفقیت تدریجی عامل در یادگیری سیاست مناسب در محیط تعیین‌شده است.



شکل ۲۳_ ۳ تغییرات پاداش ها با الگوریتم Q_learning

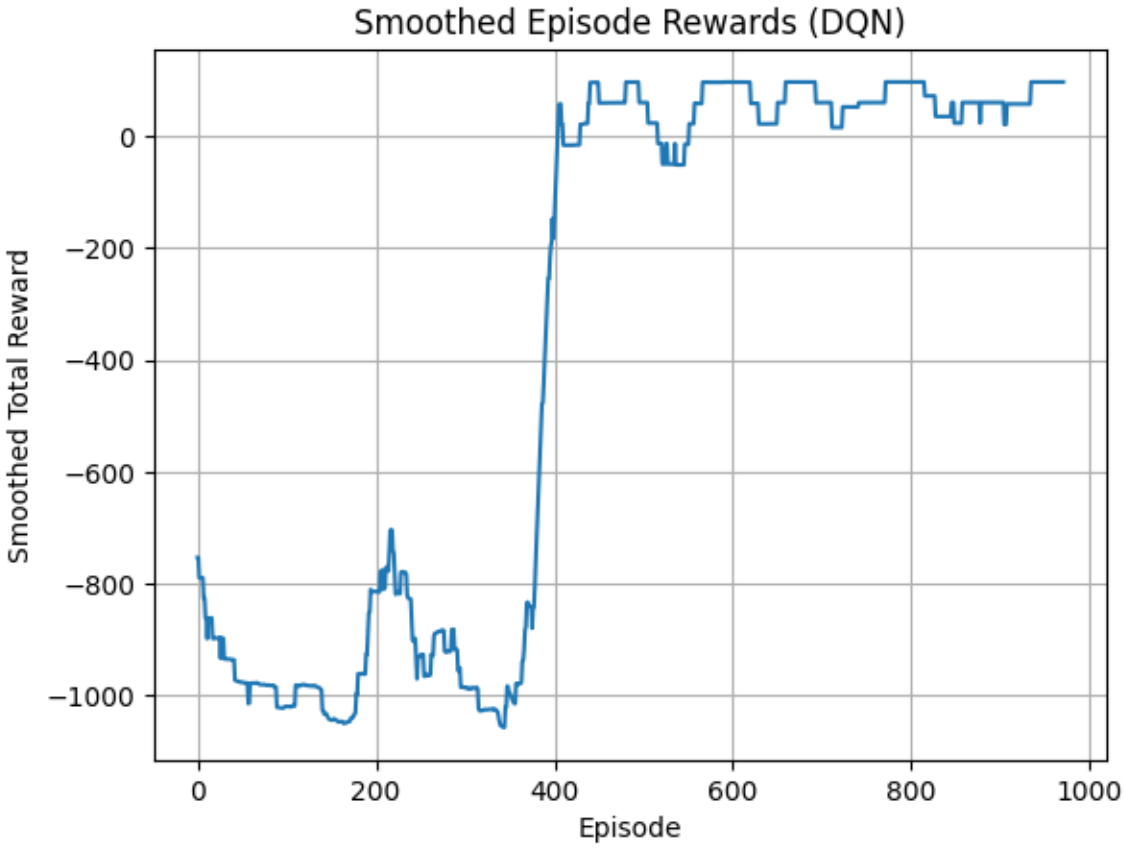
برای درک شهودی تر در این بخش از پروژه مربوط به نمایش گام‌به‌گام مسیر بهینه عامل در محیط Wumpus World میپردازیم که پس از آموزش با الگوریتم Q-Learning استخراج شده است. با استفاده از تابع show_optimal_path_visual، عامل از موقعیت شروع حرکت می‌کند و در هر مرحله، بر اساس بیشترین مقدار Q موجود در جدول یادگیری، بهترین عمل را انتخاب کرده و به موقعیت بعدی می‌رود. محیط پس از هر گام به‌صورت متنی چاپ می‌شود که موقعیت عامل با علامت مربع مشخص شده است. این نمایش پویا، امکان درک دقیق‌تر از تصمیم‌گیری عامل و بررسی کیفیت مسیر بهینه را فراهم می‌کند.



شکل ۲۴_۳ نمایش بصری نتیجه آموزش Q_learning

در بخش بعدی، برای آموزش عامل هوشمند در محیط Wumpus World از الگوریتم یادگیری تقویتی Deep Q-Learning (DQL) استفاده شد. برخلاف نسخه‌ی کلاسیک Q-Learning که از جدول Q برای ذخیره مقادیر ارزش استفاده می‌کند، در این روش از شبکه‌ی عصبی برای تخمین مقادیر Q بهره گرفته شده است. ورودی شبکه به‌صورت یک بردار One-Hot با ابعاد ۱۶ = ۴×۴ طراحی شد که نشان‌دهنده‌ی موقعیت عامل روی محیط گریدی Wumpus است. خروجی شبکه شامل ۴ مقدار Q برای چهار عمل «بالا، پایین، چپ و راست» است. مدل شبکه شامل دو لایه Fully Connected با ۶۴ نورون مخفی و تابع فعال‌ساز ReLU است.

در طول آموزش، از حافظه‌ی بازپخش (Replay Memory) برای ذخیره‌ی تجربه‌ها و یادگیری از نمونه‌های تصادفی استفاده شد. سیاست انتخاب عمل نیز به صورت ε-greedy پیاده‌سازی شد تا عامل در ابتدا بیشتر به اکتشاف بپردازد و به‌تدریج با کاهش مقدار ε به بهره‌برداری روی بیاورد. پارامترهای آموزشی شامل ۱۰۰۰ اپیزود، نرخ یادگیری ۰.۰۰۱، ضریب تخفیف γ=0.95 و نرخ کاهش ε برابر با ۰.۹۹۵ در هر اپیزود بودند. در نهایت، نمودار پاداش تجمعی اپیزودها با اعمال هموارسازی متحرک ترسیم شد تا روند یادگیری بهتر قابل مشاهده باشد.



شکل ۲۵_۳ تغییرات پاداش ها با الگوریتم DQL

نمودار پاداش تجمعی (Smoothed Episode Rewards) نشان‌دهنده‌ی بهبود قابل‌توجه یادگیری عامل پس از حدود ۴۰۰ اپیزود است. در ابتدای آموزش، عامل عمدتاً عملکرد ضعیفی دارد و پاداش‌ها عمدتاً منفی و حتی نزدیک به -۱۰۰۰ هستند که نشان‌دهنده‌ی افتادن در چاله یا برخورد با وامپوس در اپیزودهای اولیه است. اما پس از گذشت تقریباً ۳۰۰ تا ۴۰۰ اپیزود، عامل به‌تدریج استراتژی بهینه‌تری را یاد می‌گیرد و پاداش‌ها با یک شیب تند رو به افزایش می‌روند. در ادامه، عملکرد عامل به حالتی باثبات و مثبت می‌رسد، به‌طوری که پاداش‌ها در بازه‌ی نسبتاً ثابتی باقی می‌مانند و نوسان آن‌ها کاهش یافته است. این رفتار نشان می‌دهد که عامل مسیر بهینه برای رسیدن به طلا و پرهیز از خطرها را یاد گرفته و در حال بهره‌برداری از دانش اکتسابی خود است.
این روند تأیید می‌کند که الگوریتم DQN با تنظیمات مناسب و استفاده از شبکه عصبی عمیق توانسته ساختار محیط Wumpus World را به‌درستی مدل کند و با موفقیت عامل را آموزش دهد.

ب .عملکرد Policy

پرسش اول :

در بررسی نحوه‌ی عملکرد عامل با استفاده از الگوریتم‌های Q-learning وDQN ، دو نکته اصلی مشاهده شد: نخست، با رسم پاداش تجمعی در طول اپیزودها، مشخص شد که در هر دو روش، عامل با یادگیری تدریجی توانسته است عملکرد خود را بهبود دهد. درQ-learning ، بهبود پاداش‌ها نسبتاً آهسته و با نوسانات بیشتر صورت گرفت، در حالی که درDQN ، بهبود به‌صورت ناگهانی و پس از حدود اپیزود ۴۰۰ مشاهده شد و پاداش‌ها به محدوده‌ای پایدارتر و نزدیک به صفر متمایل شدند، که نشان‌دهنده‌ی شکل‌گیری سیاست بهینه پایدار در این روش است.

پرسش دوم :

در مقایسه‌ی میانگین پاداش‌ها پس از ۱۰۰۰ اپیزود، نتایج عددی نیز مؤید همین تحلیل است. میانگین پاداش در Q-learning برابر با 356.58- و در DQN برابر با 343.40- به دست آمد. گرچه اختلاف میانگین‌ها زیاد نیست، اما مقدار بالاتر میانگین پاداش درDQN ، همراه با رفتار پایدارتر در نمودار، نشان می‌دهد که DQN در این مسئله عملکرد بهتری نسبت به Q-learning کلاسیک داشته است. این برتری می‌تواند ناشی از توانایی بالاتر شبکه‌های عصبی در تقریب توابع ارزش و استخراج ویژگی‌های مؤثر باشد.

```
Average Reward (Q-learning): -356.58
Average Reward (DQN): -343.40
```

شکل ۲۶_۳ میانگین پاداش تجمعی QL , DQL

ج)

در ابتدای آموزش، مقدار اپسیلون به‌صورت عمدی بالا در نظر گرفته شده است($\varepsilon \approx 1$) ، تا عامل بتواند محیط را بیشتر کاوش کند و از انجام اکتشافات (exploration) مختلف، اطلاعات کافی درباره پاداش‌های نواحی مختلف محیط به دست آورد. در این بازه، همان‌طور که در نمودارها قابل مشاهده است، عملکرد عامل نوسانی و با پاداش منفی زیاد همراه است، چون عامل در حال آزمون و خطا و یادگیری تجربی است.

با گذشت زمان و کاهش تدریجی اپسیلون به سمت مقدار نهایی($\varepsilon \to 0.01$) ، عامل کمتر رفتار تصادفی و بیشتر رفتار آموخته‌شده را دنبال می‌کند .در نمودار هر دو الگوریتم، پس از حدود اپیزود ۳۵۰ تا ۴۰۰، رشد چشمگیر پاداش تجمعی دیده می‌شود که نشان می‌دهد عامل شروع به استفاده از دانسته‌های خود کرده و استراتژی بهینه‌تری را پیاده‌سازی کرده است.

به‌طور خاص در الگوریتمDQN ، کاهش ناگهانی و به‌دنبال آن افزایش سریع پاداش از اپیزود ۳۸۰ به بعد، بیانگر این است که شبکه عصبی پس از جمع‌آوری داده کافی و کاهش اپسیلون، الگوی بهینه‌ای برای حرکت در محیط یاد گرفته و از آن بهره می‌برد. بنابراین، کاهش اپسیلون در زمان مناسب، عامل را از فاز یادگیری تجربی به سمت بهره‌برداری مؤثر هدایت می‌کند .در غیر این صورت، باقی‌ماندن اپسیلون در سطح بالا باعث تداوم حرکات تصادفی و عدم تثبیت رفتار عامل خواهد شد.

د)

پرسش اول :

برای بررسی اینکه در چه زمانی عامل Q-learning توانسته به‌طور پایدار یاد بگیرد که چگونه طلا را پیدا کند بدون اینکه در تله بیفتد یا توسط ومپوس خورده شود، یک رویکرد ساده و کاربردی پیاده‌سازی کردیم. در این روش، ما لیست پاداش‌های به‌دست‌آمده در طول اپیزودها را بررسی می‌کنیم و دنبال اولین نقطه‌ای می‌گردیم که در آن عامل برای چند اپیزود متوالی (مثلاً ۵ بار پشت سر هم) موفق به گرفتن پاداش کامل طلا شود. چنین الگویی می‌تواند نشانه‌ای از یادگیری پایدار باشد، چرا که احتمال رسیدن تصادفی به طلا بدون شکست به‌صورت پشت سر هم بسیار پایین است. برای این کار از حلقه‌ای استفاده کردیم که روی لیست پاداش‌ها حرکت کرده و هرجایی که ۵ مقدار پشت سر هم برابر با ۱۰۰ (پاداش رسیدن به طلا) باشند را شناسایی کند. اگر چنین الگویی پیدا شود، شماره اپیزود آن ثبت می‌شود و در نهایت چاپ می‌گردد.

```
EPISODES TOOK TO CONSISTENTLY REACH THE GOLD WITHOUT FAILURE = 320
```

پرسش دوم :

در مقایسه میان الگوریتم Q-Learning و Deep Q-Network (DQN) از منظر سرعت یادگیری سیاست بهینه، نتایج تجربی نشان داد که عامل Q-Learning در حدود ۳۶۷ اپیزود قادر بود رفتاری پایدار برای یافتن طلا بدون افتادن در گودال یا خورده شدن توسط ومپوس یاد بگیرد. در مقابل، عامل DQN با وجود توانایی تعمیم بهتر، به دلیل پیچیدگی بالاتر ناشی از معماری شبکه عصبی و نیاز به نمونه‌های بیشتری برای یادگیری، روند کندتری در تثبیت سیاست بهینه نشان داد. بنابراین، در این محیط ساده و با فضای حالت محدود، Q-Learningدر یادگیری سیاست بهینه سریع‌تر و کاراتر ظاهر شد. این نتیجه نشان می‌دهد که برای مسائل ساده با فضای حالت کوچک، الگوریتم‌های سنتی همچنان می‌توانند رقابتی‌تر از الگوریتم‌های مبتنی بر شبکه‌های عصبی عمل کنند.

ه)

در این پروژه، برای پیاده‌سازی عامل یادگیرنده با الگوریتمDQN ، از یک شبکه عصبی با معماری ساده و مؤثر استفاده شد. ساختار شبکه شامل یک لایه ورودی با ۱۶ نورون است که نمایانگر وضعیت محیط به‌صورت یک بردار one-hot از موقعیت مکانی عامل در محیط ۴×۴ می‌باشد. پس از آن، یک لایه پنهان با ۶۴ نورون و تابع فعال‌ساز ReLU قرار دارد که مسئول استخراج ویژگی‌های مهم از وضعیت فعلی و یادگیری روابط غیرخطی میان ورودی و خروجی است. در نهایت، لایه خروجی شامل ۴ نورون است که معادل تعداد اعمال ممکن در محیط (بالا، پایین، چپ، راست) بوده و مقادیر Q برای هر عمل را تخمین می‌زند. این معماری به دلیل سادگی محیط و محدود بودن فضای حالت انتخاب شده است تا ضمن داشتن سرعت آموزش بالا، توانایی یادگیری یک policy بهینه را نیز حفظ کند. استفاده از ReLU نیز به پایداری و سرعت همگرایی مدل کمک کرده است.