بسم الله الرحمن الرحیم

دانشگاه اصفهان

دانشکده مهندسی کامپیوتر

فاز اول پروژه در مبانی هوش و کاربردها

پک‌من و رگرسیون خطی

استاد درس: دکتر حسین کارشناس

دستیار استاد: پوریا صامتی

دانیال شفیعی
مهدی مهدیه
سید امیررضا نجفی

آبان ۱۴۰۳

فهرست مطالب

فصل ۱

Pacman

هدف از پیاده‌سازی این بخش از پروژه این است که با جستجو مسیری بهینه (از لحاظ فاصله طی شده و همچنین تعداد گره‌های جستجو شده) پیدا کنیم که در آن پک‌من بتواند همه‌ی غذاها را بخورد.

۱.۱ توابع جستجوی مسیر

همانطور که در پروژه گفته شد ما از چهار آلگوریتم عمق اول و عرض اول، هزینه‌یابی یکنواخت و $A^\star$ استفاده کردیم.

۱.۱.۱ تابع عمق اول

در این تابع هدف پیدا کردن مسیر خوردن اهداف به نحوی است که کمترین گره را جستجو کنیم. در این تابع الزاما مسیر پیدا شده بهترین مسیر نیست و مشاهده می‌شود پک‌من از بیراهه می‌رود. کد پیاده‌سازی این تابع:

```
q = util.Stack()
empty_action_list = []
visited = set()
q.push((problem.getStartState(), empty_action_list))
visited.add(problem.getStartState())
while not q.isEmpty():
    current_node, list_of_actions = q.pop()
    if problem.isGoalState(current_node):
        return list_of_actions
    for info in problem.getSuccessors(current_node):
        successor, action, step_cost = info
        if successor not in visited:
            new_list = list_of_actions + [action]
            q.push((successor, new_list))
            visited.add(successor)
```

۲.۱.۱ تابع عرض اول

در این تابع هدف پیدا کردن مسیر خوردن اهداف به نحوی است که بهترین مسیر را جستجو کنیم. در این تابع الزاما تعداد گره‌های جستجو شده بهینه‌ترین حالت نیست. اما بعد از اینکه کد آن اجرا شد پک‌من از کوتاه‌ترین مسیر به مقصد می‌رسد. کد پیاده‌سازی این تابع:

```
q = util.Queue()
empty_action_list = []
visited = set()
q.push((problem.getStartState(), empty_action_list))
visited.add(problem.getStartState())
while not q.isEmpty():
    current_node, list_of_actions = q.pop()
    if problem.isGoalState(current_node):
        return list_of_actions
    for info in problem.getSuccessors(current_node):
        successor, action, step_cost = info
        if successor not in visited:
            new_list = list_of_actions + [action]
            q.push((successor, new_list))
            visited.add(successor)
```

۳.۱.۱ تابع هزینه‌یابی یکنواخت

در این تابع هدف پیدا کردن مسیر خوردن اهداف به نحوی است که بهترین مسیر را جستجو کنیم به نحوی که در آن ممکن است طی کردن هر مسیر هزینه‌ی مشخصی داشته باشد. تفاوت این تابع با تابع جستجوی عرض اول در همین دخیل کردن هزینه‌هاست. کد پیاده‌سازی این تابع:

```
priority_queue = util.PriorityQueue()
start_state = problem.getStartState()
priority_queue.push((start_state, []), 0)
visited = set()

while not priority_queue.isEmpty():
    state, actions = priority_queue.pop()

    if problem.isGoalState(state):
        return actions

    if state not in visited:
        visited.add(state)

        successors = problem.getSuccessors(state)
        for successor, action, cost in successors:
            if successor not in visited:
```

```
                new_cost = problem.getCostOfActions(actions +
                ↪  [action])
                priority_queue.push((successor, actions + [action]),
                ↪  new_cost)
```

۴.۱.۱ تابع $A^{\star}$

در این تابع هدف پیدا کردن مسیر خوردن اهداف به نحوی است که هم بهترین مسیر را جستجو کنیم و هم تعداد گره‌های جستجو شده را کاهش دهیم. به این منظور ما هر بار یک تخمین . تفاوت این تابع با تابع جستجوی عرض اول در همین دخیل کردن هزینه‌هاست. کد پیاده‌سازی این تابع:

```
heuristic = cornersHeuristic
priority_queue = util.PriorityQueue()
start_state = problem.getStartState()
priority_queue.push((start_state, []), 0)
visited = set()

while not priority_queue.isEmpty():
    state, actions = priority_queue.pop()

    if problem.isGoalState(state):
        return actions

    if state not in visited:
        visited.add(state)

        successors = problem.getSuccessors(state)
        for successor, action, cost in successors:
            if successor not in visited:
                new_cost = problem.getCostOfActions(actions +
                ↪  [action])
                priority_queue.push((successor, actions + [action]),
                ↪  new_cost + heuristic(successor, problem))
```

که در آن تابع cornerHeuristic را بدین نحو پیاده‌سازی کرده‌ایم:

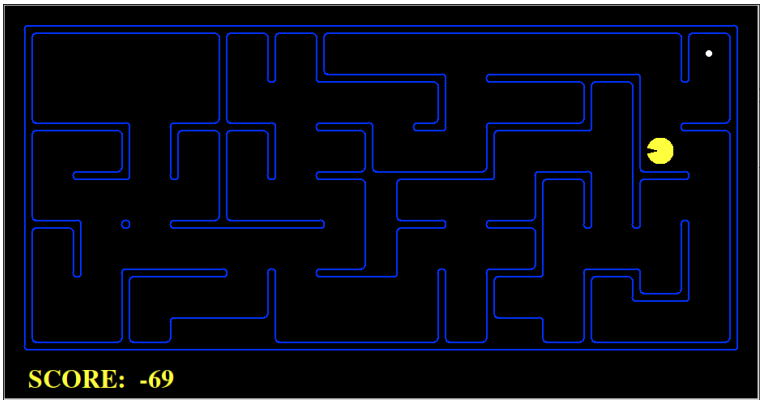
```
position, corner = state
corners = problem.corners
unseens = [corner1 for i, corner1 in enumerate(corners) if not
↪  corner[i]]

if len(unseens) == 0:
    return 0
sum1 = 0
for current_corner in unseens:
```

```
    sum1 = sum1 + util.manhattanDistance(position, current_corner)
return sum1 * len(unseens) * 2
```

باید دقت نمود این تابع یک اکتشاف غیر قابل قبول [۱] است. البته این به خودی خود ایرادی ایجاد نمی‌کند. علت اینکه تابع ما غیر قابل قبول است این است که ضرب تعداد غذاها در مجموع فاصله‌ی منهتنی می‌تواند بیشتر از فاصله‌ی واقعی باشد.
علت اینکه sum را ضربدر تعداد غذاها کردیم این بود که هر بار هنگامی که یکی از اهداف را می‌خورد، تخمین مقدار قابل توجهی کاهش می‌یافت و به نوعی به الگوریتم می‌فهماند که خوب عمل کرده است. مثلا در حالتی که پک‌من در وسط صفحه قرار داشته باشد و دو غذا در بالا چپ و پایین راست، با شرایطی که هر دو فاصله مساوی از پک‌من داشته باشند و پک‌من یکی از آنها را بخورد، تابع اکتشاف هنوز همان مقدار قبلی را برمی‌گرداند زیرا جمع فاصله‌ی منهتنی از غذاها تغییری نکرده است. اما در شرایطی که تعداد را دخیل کنیم باعث می‌شود مقدار تابع اکتشاف پس از خوردن یک هدف کاهش پیدا کند.

۲.۱ خروجی توابع

در این بخش خروجی هر تابع را در محیط‌های مختلف گزارش می‌کنیم.



شکل ۱.۱: محیط گرافیکی بازی پک‌من

۱.۲.۱ محیط ساده

```
#DFS
[SearchAgent] using function dfs
[SearchAgent] using problem type CornersProblem
Path found with total cost of 32 in 0.0 seconds
Search nodes expanded: 185
Pacman emerges victorious! Score: 478
Average Score: 0.478
Scores:        0.478
Win Rate:      1/1 (00.1)
```

[۱] Non-Adimissible

```
Record:        Win

#BFS
[SearchAgent] using function bfs
[SearchAgent] using problem type CornersProblem
Path found with total cost of 20 in 0.0 seconds
Search nodes expanded: 225
Pacman emerges victorious! Score: 490
Average Score: 0.490
Scores:        0.490
Win Rate:      1/1 (00.1)
Record:        Win

#UCS
[SearchAgent] using function ucs
[SearchAgent] using problem type CornersProblem
Path found with total cost of 20 in 0.0 seconds
Search nodes expanded: 225
Pacman emerges victorious! Score: 490
Average Score: 0.490
Scores:        0.490
Win Rate:      1/1 (00.1)
Record:        Win

#ASTAR
[SearchAgent] using function astar and heuristic nullHeuristic
[SearchAgent] using problem type CornersProblem
Path found with total cost of 20 in 0.0 seconds
Search nodes expanded: 23
Pacman emerges victorious! Score: 490
Average Score: 0.490
Scores:        0.490
Win Rate:      1/1 (00.1)
Record:        Win
```

۲.۲.۱ محیط سخت

```
#DFS
Corner -p SearchAgent  -a fn=dfs,prob=CornersProblem
[SearchAgent] using function dfs
[SearchAgent] using problem type CornersProblem
Path found with total cost of 201 in 0.0 seconds
Search nodes expanded: 340
Pacman emerges victorious! Score: 339
Average Score: 0.339
Scores:        0.339
Win Rate:      1/1 (00.1)
Record:        Win
```

```
#BFS
[SearchAgent] using function bfs
[SearchAgent] using problem type CornersProblem
Path found with total cost of 106 in 0.0 seconds
Search nodes expanded: 1966
Pacman emerges victorious! Score: 434
Average Score: 0.434
Scores:        0.434
Win Rate:      1/1 (00.1)
Record:        Win

#UCS
[SearchAgent] using function ucs
[SearchAgent] using problem type CornersProblem
Path found with total cost of 106 in 2.0 seconds
Search nodes expanded: 1966
Pacman emerges victorious! Score: 434
Average Score: 0.434
Scores:        0.434
Win Rate:      1/1 (00.1)
Record:        Win

#ASTAR
[SearchAgent] using function astar and heuristic nullHeuristic
[SearchAgent] using problem type CornersProblem
Path found with total cost of 106 in 0.0 seconds
Search nodes expanded: 187
Pacman emerges victorious! Score: 434
Average Score: 0.434
Scores:        0.434
Win Rate:      1/1 (00.1)
Record:        Win
```

۳.۲.۱ محیط بزرگ

```
#DFS
[SearchAgent] using function dfs
[SearchAgent] using problem type CornersProblem
Path found with total cost of 280 in 0.0 seconds
Search nodes expanded: 900
Pacman emerges victorious! Score: 270
Average Score: 0.270
Scores:        0.270
Win Rate:      1/1 (00.1)
Record:        Win
```

```
#BFS
[SearchAgent] using function bfs
[SearchAgent] using problem type CornersProblem
Path found with total cost of 210 in 1.0 seconds
Search nodes expanded: 11392
Pacman emerges victorious! Score: 340
Average Score: 0.340
Scores:        0.340
Win Rate:      1/1 (00.1)
Record:        Win

#UCS
[SearchAgent] using function ucs
[SearchAgent] using problem type CornersProblem
Path found with total cost of 210 in 7.1 seconds
Search nodes expanded: 11392
Pacman emerges victorious! Score: 340
Average Score: 0.340
Scores:        0.340
Win Rate:      1/1 (00.1)
Record:        Win

#ASTAR
[SearchAgent] using function astar and heuristic nullHeuristic
[SearchAgent] using problem type CornersProblem
Path found with total cost of 286 in 1.0 seconds
Search nodes expanded: 969
Pacman emerges victorious! Score: 264
Average Score: 0.264
Scores:        0.264
Win Rate:      1/1 (00.1)
Record:        Win
```

توضیح: نتیجه امتیاز و تعداد گره‌های جستجو شده تابع هزینه‌یابی یکنواخت با تابع جستجوی عرض اول یکی است! فقط هزینه‌یابی در زمان جستجو بیشتر است. علت آن این است که هزینه‌ی حرکت به خانه ۱ واحد درنظر گرفته شده و چون تابع هزینه‌یابی یکنواخت روی هزینه‌ها متمرکز است از قضا زمان بسیار بیشتری هم طول کشیده! واضح است که ممکن است در شرایطی که هزینه‌ی رفتن به هر خانه برابر نباشد نتایج متفاوت شوند.

فصل ۲

رگرسیون خطی

در این بخش از پروژه ما قصد داشتیم با داده‌های آموز مدلی را بسازیم که با آن داده‌های جدید را پیش‌بینی کنیم.

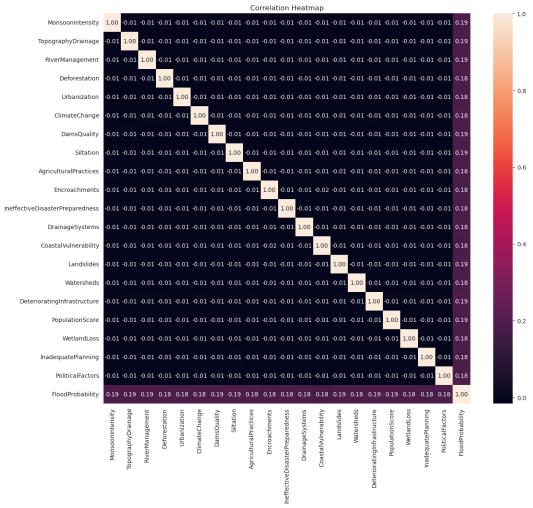
۱.۲ اکتشاف درون داده‌ها

با توجه به اینکه همه‌ی ستون‌های ما به جز id متغیرهای پیوسته بودند، ما می‌توانستیم به خوبی خود احتمال را که یک چیز پیوسته است پیش‌بینی کنیم. صرفا یک شک درمورد این وجود داشت که هاid در میان سطرهای یکی باشند. برای همین این را بررسی کردیم و چنین نبود. بنابرین متغیر id را که یک متغیر categorical بود حذف کردیم.

۱.۱.۲ پر کردن مقادیر خالی

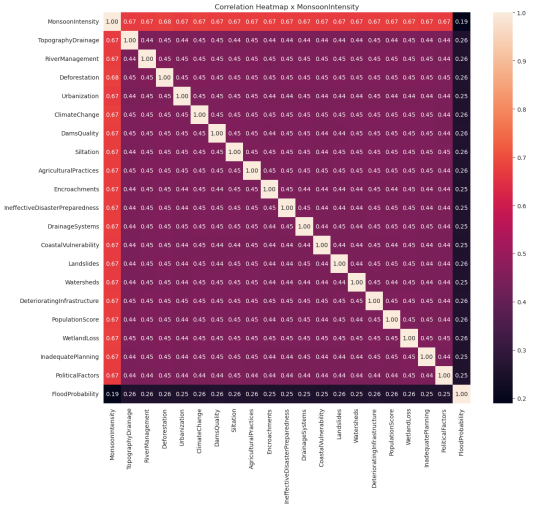
داده‌ها هیچ مقادیر خالی نداشتند اما ما یک تابع knn imputer نوشتیم که به‌جای میانگین گرفتن از کل داده‌ها از k همسایه نزدیک (با توجه به بقیه‌ی ویژگی‌ها) میانگین می‌گیرد و مقادیر را پر می‌کند. فاصله در این تابع به صورت فاصله‌ی اقلیدسی تعریف شده است. در نهایت اما به علت پیچیدگی ما از یک تابع میانگین ساده روی کل مقادیر برای پر کردن داده‌ها استفاده کردیم.

۲.۱.۲ رسم نقشه گرمایی همبستگی



شکل ۱.۲: نقشه گرمایی همبستگی متغیرها

تقریبا هیچ همبستگی بین داده‌ها وجود ندارد تا آن را حذف کنیم. به عنوان یک پیشنهاد ما تست کردیم که اگر دو ستون را در هم ضرب کنیم چه اتفاقی برای همبستگی و همچنین میانگین مربع خطا می‌افتد برای همین در نوتبوک feature.ipynb این قضیه را تست کردیم. این کار را برای همه‌ی ستون‌ها تکرار کردیم و نتیجه همبستگی مقداری بهتر شد. بنابرین ما یک تابع جدید با ضرب بالاترین میانگین همبستگی ایجاد کردیم که البته در ادامه از این کار پشیمان شدیم! (چون در همه‌ی شرایط بدتر عمل می‌کرد)



شکل ۲.۲: نقشه گرمایی همبستگی متغیرها در صورت ضرب ستون MonsoonIntensity در همه‌ی ستون‌های دیگر به‌جز خودش و FloodProbability

۳.۱.۲ اسکیل کردن

در این قسمت ما از اسکیلر استاندارد و اسکیلر کمینه بیشینه استفاده کردیم.

۴.۱.۲ آماده‌سازی داده‌ها برای آموزش مدل

داده‌ها را به صورت ۴ مجموعه داده در حالت‌های مختلف با اسکیلرهای مختلف آماده کردیم و متغیرهای وابسته‌ی آن را از متغیر مستقل آن جدا کردیم. سپس یک بردار تصادفی از وزن‌ها (slope) و یک بردار تصادفی از مقادیر ثابت (intercept) آماده کردیم تا نتایج حاصل از مدل را در آن بریزیم.

۲.۲ مدل‌سازی داده‌ها و آموزش

در این قسمت ما به مدل‌سازی داده‌ها و پیش‌بینی می‌پردازیم.

۱.۲.۲ تابع رگرسیون خطی

تابع هزینه این تابع در اول کدنویسی بسیار ساده بود اما با اضافه شدن سایر ویژگی‌ها کمی به پیچیدگی آن افزوده شد. در این تابع تعدادی ایپاک از کاربر دریافت می‌شود. سپس مشتق خطای هر سطر نسبت به مقدار واقعی محاسبه می‌شود. تابع خطای ما در اینجا مشتق تابع میانگین مربع خطاست که ما را به نقطه‌ی بهینه هدایت می‌کند.

تکانه ما در اینجا یک velocity تعریف کردیم که برای اینکه تکانه واقعا تغییر کند، در اینجا به‌جای اینکه خطا مستقیما روی slope اثر بگذارد (با learning rate مشخص)، از سرعت قبلی هم کاملا اثر می‌پذیرد و این کمک می‌کند از داده‌های outlier کمتر اثربپذیریم و سریع‌تر به چیزی که می‌خواهیم میل کنیم.برای مثال با تکانه کم، ما حدود ۶۰۰ تکرار نیاز داشتیم تا زودهنگام به نتیجه برسیم اما با پیاده‌سازی تکانه این عدد به ۳۴۰ هزار رسید و زمان را به نصف کاهش داد!

توقف زودهنگام مسئله‌ی بعدی که قبل‌تر هم به آن اشاره شد خروج زود هنگام است. قبل از پیاده‌سازی این عملکرد، زمان اجرای یک دور تابع جبرخطی برای ۴ مجموعه داده حتی با وجود اینکه نمودارها جدا رسم می‌شدند برای ۱۰ ایپاک چیزی حدود ۶ دقیقه و نیم بود! اما با پیاده‌سازی این تابع، با حفظ عملکرد، تابع آموزش در کمتر از ۱۰ ثانیه هر ۴ مدل را آموزش می‌داد!

عادی‌سازی هرچند تکنیک عادی‌سازی L2 در کد پیاده‌سازی شد که باعث می‌شود خودگردیان با مقادیر قبلی slope جمع شود ولی نتیجه‌های این اتفاق باعث می‌شد توابع underfit شود و اصلا به چیزی که می‌خواهیم شبیه نشود. یعنی با پیاده‌سازی این مکانیسم، امتیاز R^2 روی داده‌های تست به کمتر از ۰٫۸ می‌رسید که این از نظر ما مطلوب نبود. هرچند در کارکرد فلسفه‌ی آن هم همین است.

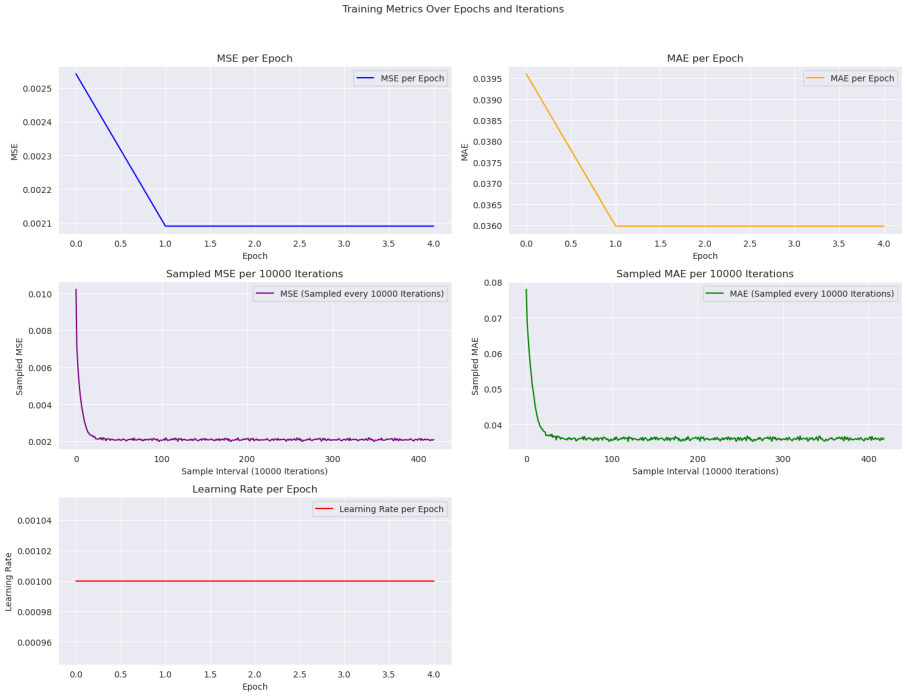
تغییر سرعت یادگیری ما از یک تکنیک ساده استفاده کردیم که در آن نرخ یادگیری بعد از هر ایپاک در یک مقدار کمتر از ۱ ضرب می‌شود. این با این فرض درنظر گرفته شده که مدل از جایی به بعد همگرا می‌شود. بیش از اندازه کوچک بودن این مقدار باعث می‌شود مدل underfit شود.

نمونه‌برداری سیاست ما این بود به‌جای مقایسه‌ی ایپاک‌ها برای توقف زودهنگام و همچنین به‌جای رسم همه‌ی تکرارها از نمونه‌برداری بین تعدادی سطر استفاده کنیم. بدین منظور میانگین خطای هر مثلا ۱۰۰۰ خط را اندازه‌گیری می‌کردیم و سپس بعد هر هزار بار آن را صفر می‌کردیم.

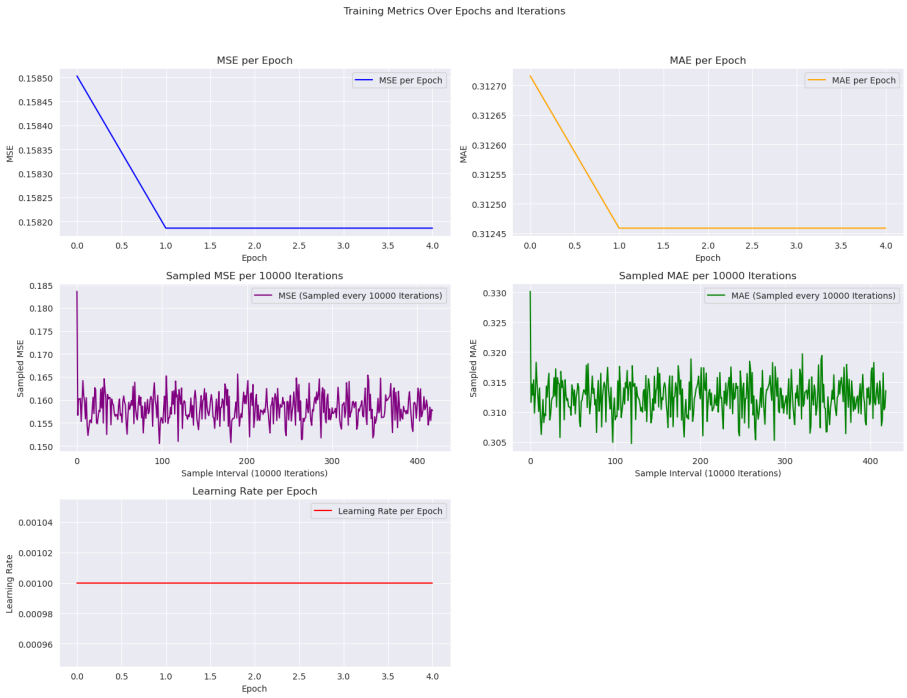
رسم تغییرات در اینجا ما کلیه تغییرات را بر حسب ایپاک و تکرار همچنین تغییرات سرعت یادگیری و خطا را نشان داده‌ایم.

۳.۲ پیش‌بینی

در این بخش، داده‌های آموزش و آزمون را به مدل می‌دهیم. تبعا به دنبال مدلی هستیم که روی هر دو داده‌ی آموزش و آزمون مقدار R^2 کافی برای ما فراهم کند.



شکل ۳.۲: آموزش داده‌های اسکیل شده با کمینه بیشینه در حالت پایه با ۵ ایپاک و سرعت آموزش ۰٫۰۱



شکل ۴.۲: آموزش داده‌های اسکیل شده با z در حالت پایه با ۵ اپاک

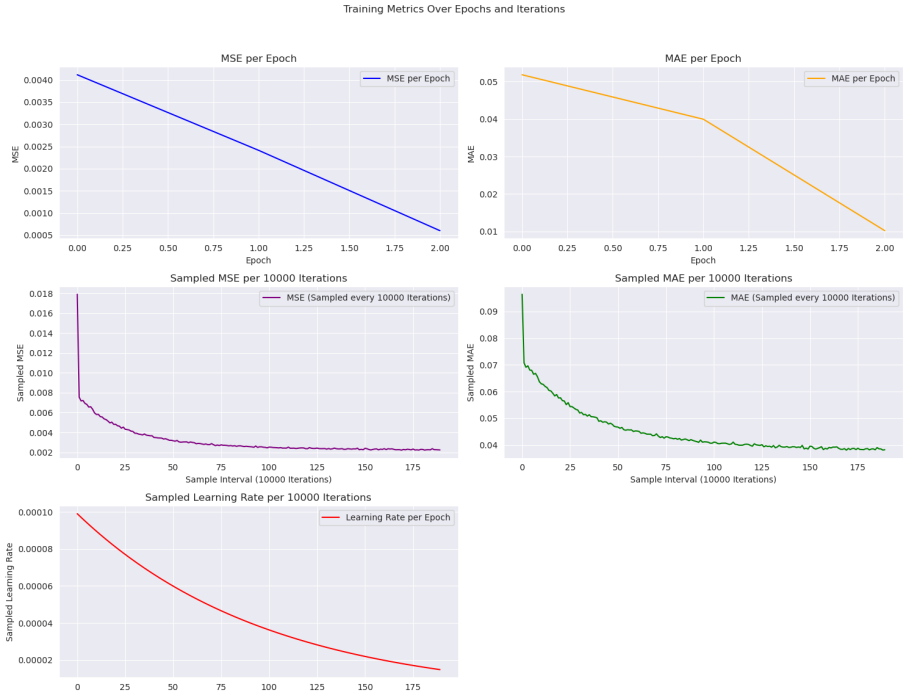
۱.۳.۲ پیش‌بینی روی داده‌های تست

برای داده‌های تست فرآیند مشابهی در زمینه‌ی اسکیل کردن طی می‌کنیم. در اینجا باید اشاره کنیم از همان اسکیلرهای آموزش استفاده می‌کنیم. سپس این داده‌ها را به مدل‌ها می‌دهیم و پیش‌بینی آن‌ها را می‌سنجیم. خروجی:

```
Mean Absolute Error: 31429430780462186.0
Mean Squared Error: 15990997244596597.0
R2 Score: 839718589887849.0

Mean Absolute Error: 036034161275208226.0
Mean Squared Error: 0020939675495502044.0
R2 Score: 8440392259660622.0

Mean Absolute Error: 3122935770427608.0
Mean Squared Error: 15784894281081954.0
R2 Score: 8417844068669359.0

Mean Absolute Error: 03595022487165275.0
Mean Squared Error: 002089504719597371.0
R2 Score: 8443716219547084.0
```

۲.۳.۲ بازی کردن با پارامترها

در نهایت ما می‌توانیم با بازی کردن با پارمترهای تابع رگرسیون در زمان کوتاه به توابع بسیار خوبی برسیم. نمونه‌ی این تابع:



شکل ۵.۲: Early stopping at iteration 2370000 with MSE: 0.002220466560191889

که در آن پارامترها به نحو زیر تعیین شده بودند:

```
epochs_number=10
initial_learning_rate=0.0001
momentum=0.5
patience=20
regularization_param=0.0
lr_decrease=0.99
iteration_sample=10000
```

که درظرف ۱۵ ثانیه آموزش مدل به نتایج زیر رسیدیم:

```
Mean Absolute Error: 0.0381413675044361
Mean Squared Error: 0.0022375861525931085
R2 Score: 0.8333423703719746
```