به نام خدا

Machine learning

گزارش تمرین 1:

Function Approximation (Snake Game)

تاریخ تحویل:

1401/01/16

فاطمه حسن زاده

40032217

امیرحسن امیرماهانی

40032722

مقدمه:

در این پروژه، ما از روش توصیف شده میچل(Mitchel) برای پیاده سازی الگوریتمی برای یادگیری بازی مار استفاده می کنیم. همانطور که می دانید، چنین وظایفی به عنوان یک نمودار انتزاعی هستند که هر حالت(state) تابلو یک گره و هر حرکت یک یال است.

مار (بازیکن) سعی می کند با سر مار به آیتم غذا را بخورد. هر آیتمی که خورده می شود مار را طولانی تر می کند، بنابراین اجتناب از برخورد با مار به تدریج دشوارتر می شود. زمانی که مار به حاشیه صفحه نمایش یا خودش برخورد کند، بازیکن بازنده است.

هدف این پروژه این است که بهترین حرکت را برای هر حالت پیدا کنید که به مار کمک می کند آیتم های بیشتری بخورد در حالی که حرکات کمتری مصرف می کند.

فایل Training :

این فایل شامل کد قسمت train کردن بازی است .

تعریف:

طبق روش توصیف شده میچل(Mitchel) در این قسمت الگوریتمی را پیاده سازی می‌کنیم که بازی مار (صفحه 20×20) را با روش function approximation (تخمین تابع) یاد می‌گیرد.

ما یک تابع V تعریف می‌کنیم: $S \rightarrow \mathbb{R}$ که به هر حالت (State) یک مقدار اختصاص می‌دهد و بر اساس V بهترین حرکت را برای هر حالت پیدا می کند. اگر بتوانیم V را پیدا کنیم، مشکل حل می شود، اما فقط چند جفت <s، $V(s)$ > موجود است.

بسیاری از حالت ها مقدار V را ندارند. بنابراین باید این تابع را فقط با این نکات (مثال های training) تخمین بزنیم. به دلیل ساختار نمودار مسئله، این امکان وجود دارد که مقدار حالت‌های با مقدار مجهول را به طور تقریبی تخمین بزنیم، و می‌توان از مقادیر تقریبی برای یافتن تقریب‌های بهتر استفاده کرد. بنابراین برای حالت های با مقدار ناشناخته، باید از مقدار تقریبی حالت جانشین(successor) استفاده می‌کنیم.

به طور کلی برای پیاده سازی این الگوریتم به صورت زیر عمل می کنیم:

1. نمایش خطی را برای $\hat{V}$ انتخاب می کنیم: $\hat{V}(\vec{x}) = \hat{w}^T \vec{x}$.
2. چند ویژگی (features) خوب ($\vec{x}$) برای توصیف حالات بازی Snake پیدا می‌کنیم.
3. همه ضرایب ($\hat{w}$) را در $\hat{V}$ با صفر مقداردهی می‌کنیم.
4. در هر حالت غیر پایانی (St) حرکت خود را بر اساس $\hat{V}$ به صورت زیر انتخاب می‌کنیم:
   از بین حرکاتی که مار می تواند در وضعیت فعلی انجام دهد، حرکتی را انتخاب می‌کنیم که به بالاترین مقدار حالت (ارزش جانشینان وضعیت فعلی) منجر می شود.
   این حالت جانشین انتخابی را St+1 می نامیم وVtrain (St)= $\hat{V}(St+1)$.
   اگر St حالت پایانی است، آنگاه Vtrain $(St) = \hat{V}(S)$.
5. با توجه به حالت مورد نظر مقدار حالت را تنظیم می‌کنیم:
   به طور مثال برای حالتی که برد صورت میگیرد: Vtrain (St)=1000
6. حال ما یک نمونه جدید داریم: <St, Vtrain (st)>.
   بر اساس این نمونه به روز رسانی $\hat{w}$ به شرح زیر است:

$$wi = wi + \alpha\ (V\text{train}\ (st) - \hat{V}(st).)\ xi$$

یک مقدار کوچک برای α اختصاص می‌دهیم. زمانی که الگوریتم را train کردیم و وزن ها را پیدا کردیم($wi\ s$)، از این وزن ها برای اجرای دو عامل برای بازی با یکدیگر استفاده می‌کنیم.

پیاده سازی:

در این کد ازکلاس SnakeGame که در فایل Border تعریف کردیم،استفاده می‌کنیم.
در فایلborder ، صفحه بازی، مار، موانع(obstacles)، حرکت مار، تولیدغذا به صورت رندوم و برخی توابع دیگر را تعریف کردیم که بعدا به شرح آن ها می‌پردازیم.

به طور کلی در یک حلقه for به تعداد epoch هایی که تعریف کردیم بازی را انجام می دهیم و تا زمانی که gameover نشده، ویژگی ها (features) ، $Vtrain$ و $V(V\_h)$ را محاسبه کرده و با هر بار game over شدن در یک حلقه for دیگر با توجه به فرمولی که در مرحله 6 تعریف کردیم وزن ها را اپدیت می کنیم.

در ابتدای این حلقه کلاس SnakeGame را با طول و عرض تعریف شده صدا می‌زنیم و در متغیر game قرار می‌دهیم تا از توابع تعریف شده در این کلاس بهره ببریم.

ویژگی ها (features):

تابع calcAllFeatures() :

برای محاسبه ویژگی ها از تابع calcAllFeatures() که در کلاسSnakeGame تعریف کردیم استفاده می‌کنیم. در این تابع مختصات محلی که سر مار قرار دارد را در x وy ذخیره کرده و تابع getFeatures(x,y) را برای این مختصات و همچنین برای جهت های پایین،راست،بالاو چپ صدا می‌زنیم.

تابع getFeatures(x,y) :

در تابع getFeatures(x,y) ،ویژگی هایی که در واقع هر حالت را توصیف می‌کند ،تعریف کردیم.این ویژگی عبارت اند از:

- فاصله مار تا غذا foodDistance()
- فاصله مار تا موانع obstacleDistance()
- فاصله مار تا حاشیه صفحه بازی boardDistance()

که هر کدام ازین ویژگی ها به صورت تابع تعریف شدند.

تابع foodDistance() ، مختصات محلی که سر مار قرار دارد را در x وy ذخیره کرده و مختصات فعلی غذا را با استفاده از تابع getFoodPos() به دست اورده و در i وj قرار داده و فاصلهxوi با هم و yوj با هم را محاسبه کرده و مجموع این دو را برمی‌گرداند.

تابع ()obstacleDistance ، مختصات محلی که سر مار قرار دارد را در x وy ذخیره کرده و در دو حلقه for تو در تو مختصات موانع(بلاک 2*2) را نیز محاسبه کرده و در ارایه res قرار می دهد و سپس در یک حلقه for دیگر فاصله سر مار تا این موانع را محاسب کرده و مینیمم فاصله را برمی‌گرداند.

تابع ()boardDistance ، مختصات محلی که سر مار قرار دارد را در x وy ذخیره کرده و طول و عرض بردر را در xBorder و yBorder قرار داده و فاصله x و xBorder را محاسبه کرده در dx قرار میدهد و بین این مقدار و مقدار x مینیمم را محاسبه میکند. همچنین فاصله y و yBorder را محاسبه کرده در dy قرار میدهد و بین این مقدار و مقدار y مینیمم را محاسبه میکند و در اخر مجموع dx و dy را برمی‌گرداند.

خروجی تابع ()calcAllFeatures را به ترتیب در x، down، right، up و left قرار می‌دهیم.

تابع V (V_h) :

برای محاسبه V (V_h) که درمرحله 1 تعریف کردیم($V(\vec{x}) = \hat{w}T\,\vec{x}$) از تابع V(x, w) استفاده می کنیم و این تابع را با xها یا همان ویژگی هایی که در قسمت قبل محاسبه کردیم و wهایی که در ابتدا صفر هستند و به صورت یک وکتور 4 تایی تعریف کردیم، صدا می‌زنیم.

این تابع، xها و w ها را دریافت کرده و با توجه به تعریف تابعV ، از تابع matmul از کتابخانه numpy استفاده می‌کنیم تا ضرب ماتریسی xها و wها را محاسبه کنیم.

محاسبه Vtrain :

برای محاسبه Vtrain و با توجه به مرحله 4 که باید بالاترین مقدار جانشین را برای یک حالت محاسبه کنیم، تابع V را برای هر کدام از جهات down، right، up و left صدا می‌زنیم و همه را در آرایه successors قرار می‌دهیم. و بیشترین مقدار این آرایه را درVtrain قرار می‌دهیم. البته در صورتی که به مرحله برد بازی رسیدیم مقدار 1000 را در Vtrain قرار می‌دهیم.

تابع ()win :

برد بازی را با توجه به تابع ()win تعیین میکنیم. برد وقتی اتفاق می‌افتد که کل صفحه بازی را مار پوشش دهد. برای این کار تابع ()win ،در دو حلقه تو در تو به اندازه طول و عرض صفحه بازی ،کل خانه های صفحه بازی را چک میکند و اگر همه خانه مخالف صفر بود(صفر نشان دهنده این است که آن خانه خالی است) آنگاه true برمی‌گرداند.

حرکت مار :

برای حرکت مار و همچنین چک کردن شرط while که همان game over شدن(باختن) است از تابع makeMove(direction) که در کلاس SnakeGame تعریف کردیم، استفاده می کنیم. این تابع جهت حرکت بازی رو به عنوان ورودی دریافت کرده و طول مار و حالت باخت را برمی‌گرداند.

برای محاسبه direction ، اندیس بیشترین مقدار ارایه successors را با استفاده از تابع ()argmax از کتابخانه numpy محاسبه می‌کنیم و در direction قرار می‌دهیم.

تابع makeMove(direction):

در تابع makeMove(direction) ، ابتدا چک می‌کنیم که حرکت در جهت دریافت شده معتبر است و منجر به باخت نمی‌شود. این کار را با صدا زدن تابع checkValid(direction) انجام می‌شود. اگر خروجی checkValid(direction) برابر true بود آنگاه، مختصاتی که سر مار قرار دارد را در headX وheadY ذخیره کرده و این مقدار را با مقدار بدن مار جایگزین می‌کند. سپس چک می‌کند که اگر در مسیر حرکت به غذا رسید طول مار را یکی افزایش می دهد و همچنین تابع ()spawnFood را صدا می‌زند تا به صورت رندوم غذای جدیدی تولید کند.

اگر خروجی checkValid(direction) برابر false بود آنگاه، مقدار game over را true کرده و در اخر مقدار game over و طول مار را برمی‌گردانیم.

آپدیت وزن ها :

با توجه به فرمول مرحله 6 و مقادیری که به دست اوردیم ، بعد از هر بار game over شدن ، در یک حلقه for به اندازه وکتور w ، وزن ها را اپدیت می‌کنیم و در یک فایل ذخیره می‌کنیم.

مقدار الفا را ابتدای کد یک مقدار خیلی کم مثلا 0.000001 قرار میدهیم.

فایل Border :

در این فایل، صفحه بازی، مار، موانع(obstacles)، حرکت مار، تولیدغذا به صورت رندوم و برخی توابع دیگر را تعریف کردیم که به توضیح آن ها می‌پردازیم.

کلاس BodyNode :

این کلاس شامل برخی از توابع اولیه است.

تابع **getPosition()** :

این تابع xوy نقطه مورد نظر را برمیگرداند.

تابع **setParent()** :

این تابع parent نقطه مورد نظر را برمیگرداند.

کلاس Snake :

این کلاس شامل برخی از توابعی است که مار را توصیف می‌کند.

تابع **newHead(x,y)**:

این تابع parent نقطه ای که سر مار قرار دارد را جایگزین سر مار می‌کنیم.

تابع **moveBodyForwards()** :

در یک حلقه while چک می‌کنیم تا زمانی که parent نقطه سر مار مخالف None است آنگاه مختصات parent را جایگزین نقطه فعلی می‌کنیم.

تابع **move(direction)** :

این تابع یک متغیر به نام direction به عنوان ورودی دریافت می‌کند و مختصات سر مار را در متغیر headPosition می‌ریزد و تعیین میکند که بر اساس هر مقدار 0 تا 3 متغیر direction ، مختصات بعدی سر مار چه باشد.

کلاس SnakeGame :

این کلاس شامل برخی از توابعی است که بازی مار را توصیف می‌کند.تعدادی از توابع را در قسمت train توضیح دادیم و بقیه توابع را اینجا توضیح می‌دهیم.

در ابتدا صفحه بازی را با توجه به طول و عرض مورد نظر یک وکتور با مقادیر صفر تعریف می کنیم و به هر یک از متغیرهای سر مار، بدن مار، غذا و مانع مقادیر خاصی را نسبت می‌دهیم.

همچنین کلاس Snake را با x وy رندوم صدا میزنیم و در متغیر snake میریزیم.و متغیر سر مار را در مکان رندومی از صفحه قرار می دهیم.

تابع spawnFood() :

این تابع به صورت رندوم غذا تولید میکند یعنی چک می کند مکان هایی که توسط سر مار یا بدن مار اشغال نشده اند را درون آرایه emptyCells نگه میدارد و از درون این آرایه مقداری را به صورت رندوم انتخاب کرده و مقدار آن را برابر مقدار منتسب شده به غذا قرار می‌دهد.

تابع insertObstacle(self, count=2):

این تابع به صورت رندوم مانع های 2*2 تولید می‌کند.به این صورت که ابتدا x وy ای به صورت رندوم تولید میکند و سپس 4 جهت بالا، پایین،چپ و راست این تابع را چک میکند که اگر خالی بود مقدار آن ها را برابر با مقدار منتسب شده به مانع قرار دهد.

تابع potentialPosition(self, direction) :

این تابع یک متغیر به نام direction به عنوان ورودی دریافت می‌کند و مختصات سر مار را در متغیر xو y میریزد و تعیین میکند که هر مقدار 0 تا 3 متغیر direction مشخص کننده چه جهتی از جهات بالا، پایین،چپ و راست است. و مقدار جدید x و y را برمیگرداند.

تابع checkValid(self, direction):

این تابع یک جهت دریافت می‌کند و با استفاده از تابع بالا x و y جدید سر مار را داخل دو متغیرnewX وnewY نگه میدارد و چک میکند که اگر این مختصات جدید باعث برخورد به دیوار،برخورد به مانع و برخورد به خود مار نشود آنگاه مقدار true را برمی‌گرداند.

تابع getFoodPos():

این تابع با استفاده از دو حلقه تو در تو مختصات مکانی که غذا در ان قرار دارد را برمی‌گرداند.

تابع display() :

در این تابع برای حاشیه های صفحه بازی و هر یک از متغیرهای سر مار، بدن مار، غذا و مانع کاراکتری را نسبت می دهیم و پرینت می‌کنیم.

تمرین شماره یک

امیرحسن امیرماهانی و فاطمه حسن زاده

نتیجه: