به نام خدا

دانشکده مهندسی کامپیوتر

مبانی هوش مصنوعی ترم بهار ۰۲-۰۱

پروژه سوم: الگوریتمهای تکاملی (ژنتیک)

مهلت تحویل ۷ تیر ۱۴۰۲

صورت مسئله

شرکت نینتندونت سو می در حال توسعه یک بازی است که در آن یک الگوریتم بصورت کاملا رویه ای مراحل جدیدی را می‌سازد تا بازیکنان تعداد زیادی مرحله برای بازی کردن داشته باشند. متاسفانه این الگوریتم ساخت مرحله با احتمال تقریبا کمی، مراحلی می‌سازد که غیر قابل رد کردن است و به خاطر اینکه اعتبار باقیمانده شرکت زیر سوال نرود، پیشنهاد می‌شود که یک الگوریتم دیگر طراحی شود که بتواند مراحل را بازی کند و قابل حل بودن یا غیر قابل حل بودن آن را گزارش کند.

پیشنهاد اولیه برای چنین الگوریتمی استفاده از یادگیری عمیق است؛ به این صورت که یک عامل طراحی شود که بعد از آموزش دیدن بتواند تمام مراحل را حل کند ولی به دلیل پیچیدگی مراحل و زمان زیاد یادگیری، با توجه به اینکه زمان کمی تا انتشار بازی مانده، از این راه حل صرفنظر می‌شود. روش دیگر استفاده ترکیبی از یادگیری تقویتی و یادگیری تقلیدی است که نیاز به این دارد که بازیکنان مراحل را بازی کنند تا عامل از آنها یاد بگیرد، ولی چون هنوز بازی منتشر نشده است این راه هم اکنون غیر ممکن است.

راه حل آخر، استفاده از الگوریتم های تکاملی است که برخلاف روش‌های قبلی، سعی در حفظ کردن هر مرحله دارد، ولی با سرعت خوبی به پاسخ می‌رسد و توسعه‌ی آن به زمان کمی نیاز دارد.

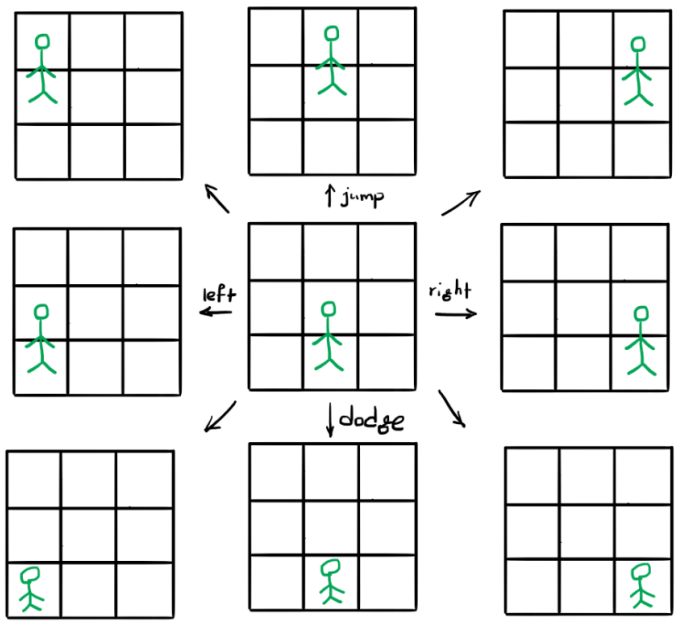
شما که اخیرا با موفقیتی که در یک شرکت ساخت میز داشتید یک شرکت برای خودتان تاسیس کرده اید، با قراردادی وظیفه پیاده سازی این الگوریتم را به عهده می‌گیرید. وظیفه شما این است که برای مراحل ساخته شده توسط الگوریتم A که مراحل ساده تولید می‌کند یک الگوریتم تکاملی توسعه دهید که تعیین کنید یک مرحله با الگوریتم شما قابل رد شدن است یا نه (واقعا ساده است یا نه) و در صورت قابل رد شدن بودن، مقدار حدودی حداکثر امتیاز را بدست آورد تا بر اساس آن به کسانی که به این امتیاز (یا احتمالا بیشتر) از آن رسیدند جایزه داده شود.

قوانین بازی:

هدف این بازی بالا بردن پرچم بدون برخورد به دشمن است. در مسیر بازی سه نوع آیتم وجود دارد:

- گومپا، اینها دشمنان زمینی هستند که نباید به آنها برخورد کرد و باید از روی آنها پرید.
- لاکیپو، اینها دشمنانی هستند که در هوا هستند و باید از زیر آنها جاخالی داد.
- قارچ، با گرفتن قارچ به امتیاز بازیکن اضافه می‌شود.

توجه شود که هیچ یک از آیتم ها حرکت نمیکنند.
در این بازی میتوان به چپ و راست رفت و پرید یا جاخالی داد. به چپ یا راست رفتن، مالوئیجی، شخصیت اصلی بازی را یک واحد به چپ یا راست می‌برد و پریدن، او را یک واحد به بالا می‌برد و جاخالی دادن او را به پایین خم می‌کند. علاوه بر این عملیات چپ/راست و پرش/جاخالی می‌توانند همزمان انجام شوند، مثلا می‌توانیم هم به راست برویم و هر بپریم.



در صورت پریدن، مالوئیجی، یک واحد زمانی در هوا می‌ماند و وقتی در هوا است دیگر نمی‌تواند بپرد و در واحد زمانی بعدی به زمین باز می‌گردد. ولی وقتی در هوا است، میتواند به سمت چپ یا راست برود و در واحد زمانی بعدی به زمین چپ یا راست برود. اما برای جاخالی می‌توان دو واحد پشت سر هم جاخالی داد.

هر مرحله بصورت یک رشته حرف نمایش داده میشود (بدون فاصله). _ به معنی زمین خالی است، G به معنی گومپا (دشمن زمینی)، L به معنی لاکیپو (دشمن هوایی) و M قارچ(امتیاز) است. مالوئیجی بازی را از اولین مکان(حرف) سمت چپ شروع می‌کند و باید به آخرین مکان برسد.


```
_ _ _ _ G _ M L _ _ G _
```

برای مثال مرحله تصویر زیر می‌تواند با رشته زیر آن نمایش داده شود:

خروجی هر عامل در هر مرحله باید شامل یک رشته باشد که هر کاراکترش، نشاندهنده یک عمل در آن موقعیت است. میدانیم همیشه مالوئیجی از چپترین نقطه شروع می‌کند و هدف راستترین نقطه است، پس برای دریافت امتیاز بهینه، نیازی به حرکت به چپ نیست، پس برای توسعه این الگوریتم عامل همیشه به راست حرکت می‌کند. پس سه اکشن می‌توانیم در هر مرحله انجام دهیم:

- حرکت به راست: 0
- پریدن و حرکت به راست: 1
- جاخالی و حرکت به راست: 2

هر کدام از این اکشن‌ها را با عدد مشخص شده در خروجی نشان می‌دهیم. برای مثال مرحله بالا به حداقل 12 اکشن برای اتمام نیاز دارد؛ پس از آنجایی که عامل ما همواره به راست حرکت می‌کند، با یک رشته 12 تایی می‌توان اکشن‌هایش را نمایش داد. یک رشته اکشن مورد انتظار برای مرحله بالا به شکل زیر است:

```
000100201000
```

همانطور که دیده می‌شود، مالوئیجی با رشته اکشن بالا می‌تواند مرحله را به پایان برساند. به این شکل که مثلا قبل از رسیدن به گومپای خانه 5، در خانه 4 می‌پرد و در خانه 5 در هوا است. به این شکل در خانه 6 فرود می‌آید (چون علاوه به پرش به راست هم حرکت کرده بود). همچنین با نپریدن از خانه 6، قارچ خانه 7 را می‌گیرد. دقت کنید که مثلا اگر مالوئیجی در خانه هفت، بجای جاخالی دادن، هر کار دیگری می‌کرد، می‌سوخت!

توجه کنید که رشت‌های با 12 تا صفر هم برای بازی قابل قبول است ولی موجب پیروزی نمی‌شود.

جزئیات پیاده‌سازی

الگوریتم‌های تکاملی بصورت حلقوی در چند نسل اجرا می‌شوند. بصورت کلی الگوریتم‌های ژنتیکی شامل 5 مرحله اصلی هستند:

1. جمعیت اولیه:

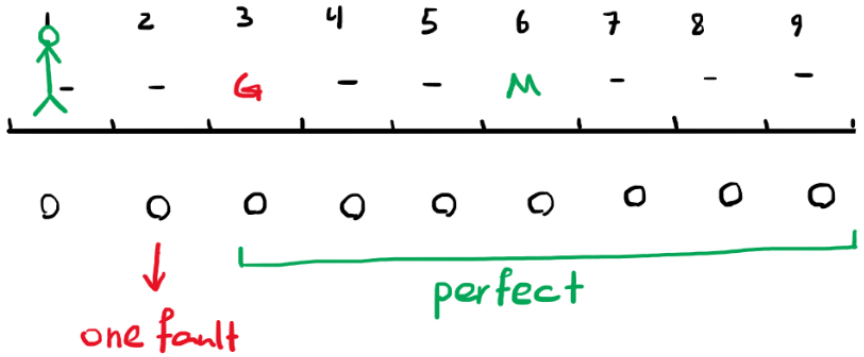
برای تولید جمعیت اولیه، به تعداد دلخواه مثلا 200 کروموزوم بسازید. هر کروموزوم نماینده یک راه حل است (مثلا همان رشته با 12 صفر). در تولید رشته‌ها کاملا آزاد هستید؛ برای مثال می‌توانید کاملا بصورت تصادفی n کاراکتر از بین 0 و 1 و 2 انتخاب کنید یا به آنها وزن بدهید.

توجه کنید که در این مرحله باید فرض کنید از خود مرحله خبر ندارید؛ یعنی مثلا نمی‌توانید بر اساس تعداد و مکان گومپاها جمعیت اولیه خود را تعیین کنید.

2. محاسبه شایستگی

اولین هدف رسیدن به پرچم است، پس تعداد گام‌های برداشته شده به جلو یا مکانی که بازیکن در آن می‌سوزد می‌تواند مهمترین عامل در محاسبه‌ی امتیاز باشد. توجه شود ممکن است یک کروموزوم(رشته اکشن) طوری باشد که اول بازی بسوزد ولی اگر در آنجا نمی‌سوخت، سایر اکشن‌هایش کاملا بهینه بودند و می‌توانست بازی را برنده بشود و در نسل بعد موثر باشد؛ به همین خاطر می‌توان شایستگی را طولانی‌ترین مسیری که بدون سوختن طی می‌شود در نظر گرفت.

برای مثال، در نمونه زیر شایستگی می‌تواند بجای 2، 6 باشد چرا که از خانه 3 تا آخر اکشن ها بهینه بودند.



به این نکته نیز توجه کنید که برنده شدن برای ما ارزش زیادی دارد، بنابراین می‌توان مقداری امتیاز (مثلا 5) به امتیاز کل کروموزومی که برنده شده است اضافه کرد؛ با اینکار مطمئن می‌شویم کروموزومی که با 10

امتیاز سوخته و کروموزومی که با 10 امتیاز برنده شده، از نظر اهمیت، با هم مساوی نخواهند بود. توجه کنید اگر خیلی به امتیاز کروموزوم برنده اضافه کنیم، هدفش فقط برنده شدن می‌شود و به سایر نکات ریز توجه‌ی نمی‌کند.

نکته‌ی دیگر در محاسبه امتیاز، در نظر گرفتن قارچ‌های خورده شده است، در این بازی هر قارچ دو امتیاز دارد پس برای محاسبه شایستگی، این را نیز باید مدنظر قرار دهید. همینطور درصورت پریدن در مکان آخر نقشه (قبل از رسیدن به پرچم) نیز یک امتیاز اضافی به بازیکن داده می‌شود.

یک نمونه‌ی ساده از تابع شایستگی قبل از توضیحات تکمیلی آورده شده است. شما می‌توانید تابع شایستگی ساده یا پیچیده‌ای داشته باشید ولی به خاطر داشته باشید که فقط رسیدن به خط پایان، مورد نظر نیست.

امتیازی:

ممکن است یک کروموزوم با پرشهای بیهوده در جایی که می‌تواند نپرد هم مرحله را با امتیاز خوبی رد کند؛ برای حل این مشکل میتوانید به پرش ها مقدار کمی (مثلا 0.5 یا 1) امتیاز منفی بدهید.

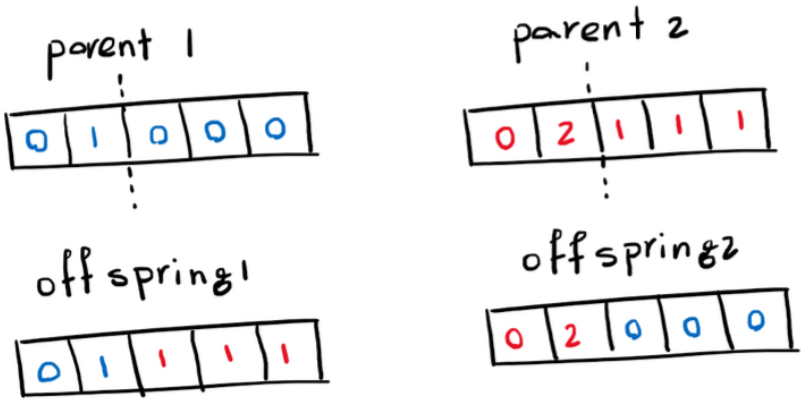
با پریدن در دو مرحله قبل از گومپاها، مالوئیجی می‌تواند روی آنها فرود بیاید و آنها را بکشد. در صورت کشتن گومپا، بازیکن 2 امتیاز اضافی می‌گیرد.

3.انتخاب

بعد از محاسبه شایستگی، باید بهترین کروموزوم ها انتخاب شوند تا برای بازترکیبی از آنها استفاده شود. برای این انتخاب می‌توانید نصف برتر جمعیت اولیه را انتخاب کنید یا بر اساس شایستگی، بصورت رندوم وزن- دار با جایگذاری، تعدادی را برای بازترکیبی انتخاب کنید.

4. بازترکیبی

در این مرحله والدین با بازترکیبی، تولید نسل جدید میکنند. جزئیات پیاده سازی این مرحله نیز آزاد است. برای مثال می‌توانید از جمعیت انتخاب شده در مرحله قبل بصورت رندوم با جایگذاری یا بدون جایگذاری والدینی را انتخاب کنید و بصورت زیر بازترکیبی را انجام دهید:



همچنین می‌توانید بصورت رندوم، برای تولید فرزند، برای هر کاراکتر، ژن یکی از والدین به ارث برده شود.

این انتخاب که آیا در نسل جدید، می‌توان از والدین نسل قبل استفاده کرد یا نه نیز به اختیار خودتان است، با این حال توصیه میشود بهترین‌های نسل قبل نگه داشته شود تا اطلاعات آنها از بین نرود؛ چون ممکن است بهترین‌های نسل قبل، فرزندان خوبی تولید نکرده باشند.

5. جهش

در نهایت در جهش، با یک احتمال کم، مثلا 0.2 می‌توانید یک یا چند تا از ژنهای یک کروموزوم را تغییر دهید. این تغییر میتواند بایاس باشد یا نباشد. برای مثال جهش می‌تواند در خدمت شایستگی باشد، به این شکل که همانطور که در قسمت محاسبه‌ی شایستگی ذکر شد، پرش‌های پی در پی خوب نیست و بخاطر همین در جهش می‌توان یک ژن را به طور تصادفی به ۰ ریست کرد که پرش‌ها کمتر شود.

الگوریتم ژنتیک نباید تا بینهایت ادامه پیدا کند، به همین خاطر نیازمند شرط پایان هستیم. برای این کار می توانید تا جایی صبر کنید که میانگین شایستگی نسل، همگرا شود (تغییر میانگین شایستگی کمتر از اپسیلون) یا مثلا تعداد ثابتی نسل مثلا 10 در نظر بگیرید. همچنین میتوان برای اطمینان از هر دو شرط گفته شده استفاده کرد.

نکته:

به خاطر داشته باشید در رشته اکشنها، قبل از اینکه مالوئیجی به گومپا برسد باید بپرد و قبل از اینکه به لاکیپو برسد باید جاخالی بدهد. از آنجا که وقتی مالوئیجی در هوا است (بعد از اینکه پرید و به راست رفت [اکشن ۱]) نمی‌تواند بپرد یا جاخالی بدهد، پس در صورتی که در قدم بعدی لاکیپو باشد، می‌سوزد. ولی اگر در قدم بعدی گومپا باشد، با پریدن روی آن نمی‌سوزد و آن را می‌کشد که در قسمت امتیازی محاسبه شایستگی ذکر شد.

در نهایت به خاطر داشته باشید که تمام پارامترهای گفته شده (مثلا جمعیت اولیه و امتیاز ها) برای اینکه یک دید کلی به شما بدهند مطرح شدند و شما می‌بایست با تغییرات ریز، کاری کنید که الگوریتم به کارایی قابل قبولی برسد.

امتیازی:

هر خلاقیت دیگری با صلاح تدریسیاران و بر اساس میزان خلاقیت، نمره اضافی دارد؛ برای مثال نمایش بازی بصورت گرافیکی یا اضافه کردن محدودیت‌ها و ویژگی‌های جدید به بازی، مثل محدودیت پریدن پشت سر هم؛ یعنی باید بعد از فرود آمدن حتما یک قدم به راست رفت تا بتوان دوباره پرید.

رسم نمودار

رسم نمودار برای دیباگ کردن کدهایی شبیه این، لازم است. مثلا میتوان با کمک نمودار میانگین شایستگی در هر نسل، دریافت که الگوریتم در اپتیموم محلی گیر افتاده است.

بعد از پایان پیاده سازی الگوریتم، مقدار میانگین، بهترین و بدترین شایستگی هر نسل را در یک نمودار رسم کنید (با کتابخانه‌های مربوط برای رسم نمودار).

با استفاده از نمودار های گفته شده، برای هر یک از 5 مورد ذکر شده در بخش قبلی، دو روش از پیاده‌سازی ها را برای level8 رسم کنید. برای این کار میتوانید پارامترهای الگوریتم را تغییر داده و با هم مقایسه کنید یا روش‌های مختلف یک مرحله را با هم مقایسه کنید. یک نمونه از پیاده‌سازی های مختلف در جدول زیر آورده شده است:

مرحله	روش اول	روش دوم
جمعیت اولیه	۲۰۰ کروموزوم	۵۰۰ کروموزوم
محاسبه شایستگی	با محاسبه امتیاز برنده شدن	بدون محاسبه امتیاز برنده شدن
انتخاب	فقط انتخاب برترین ها	انتخاب وزن دار بر اساس شایستگی
بازترکیبی	بازترکیبی یک نقطه‌ای	بازترکیبی دونقطه‌ای
جهش	احتمال جهش ۰.۱	احتمال جهش ۰.۵

در نهایت دو نمودار را همراه با کد به صورت فایل زیپ ارسال کنید.

ضمیمه 1: نمونه ساده تابع شایستگی (attachments/game.py)

می توانید قوانین بازی و امتیاز را در کلاسی شبیه به کلاس زیر پیاده‌سازی کنید.

```python
class Game:
    def __init__(self, levels):
        # Get a list of strings as levels
        # Store level length to determine if a sequence of action passes all the steps

        self.levels = levels
        self.current_level_index = -1
        self.current_level_len = 0

    def load_next_level(self):
        self.current_level_index += 1
        self.current_level_len = len(self.levels[self.current_level_index])

    def get_score(self, actions):
        # Get an action sequence and determine the steps taken/score
        # Return a tuple, the first one indicates if these actions result in victory
        # and the second one shows the steps taken

        current_level = self.levels[self.current_level_index]
        steps = 0
        for i in range(self.current_level_len - 1):
            current_step = current_level[i]
            if (current_step == '_'):
                steps += 1
            elif (current_step == 'G' and actions[i - 1] == '1'):
                steps += 1
            elif (current_step == 'L' and actions[i - 1] == '2'):
                steps += 1
            else:
                break
        return steps == self.current_level_len - 1, steps

g = Game(["____G__L__", "___G_M___L_"])
g.load_next_level()

# This outputs (False, 4)
print(g.get_score("0000000000"))
```

ضمیمه 2: مراحل بازی (attachments/levels/)

چند مرحله آماده با فرمت txt ضمیمه شده است که باید الگوریتم خود را روی آنها اجرا کنید.

توضیحات تکمیلی

• این پروژه را بصورت انفرادی انجام دهید.

• پروژه تحویل مجازی دارد و بخشی از نمره به تسلط به کد اختصاص دارد.

• زبان انجام پروژه آزاد است.

• کد و نمودارهای رسم شده را با هم به صورت فایل زیپ آپلود کنید.

• در صورت هرگونه سوال یا مشکل با ایمیل درس یا تدریسیاران در تلگرام در تعامل باشید.

ددلاین این پروژه 7 تیر 1402 ساعت 23:59 است.