دانشگاه صنعتی امیرکبیر

دانشکده مهندسی کامپیوتر

پروژه سوم

مبانی هوش محاسباتی

(بازی تکامل عصبی)

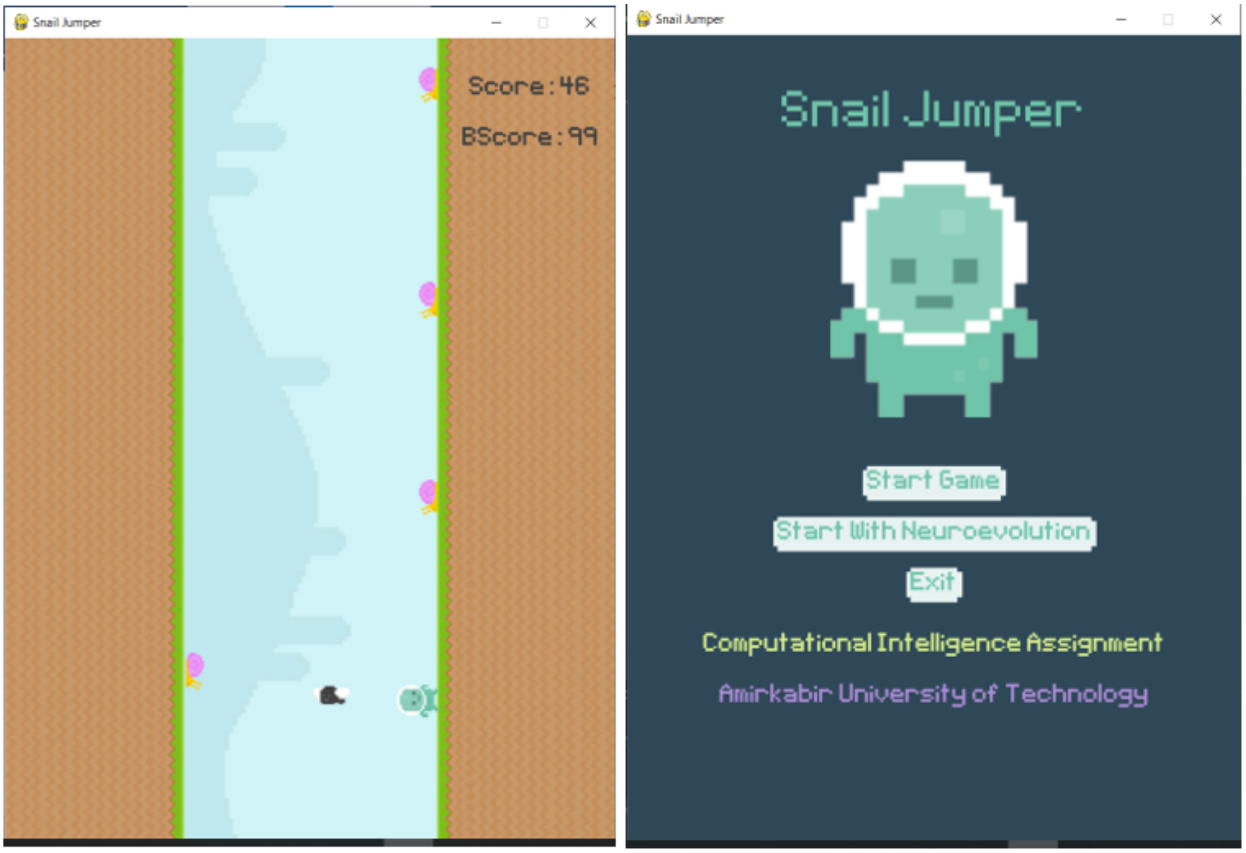
استاد درس: دکتر عبادزاده

بهار ۱۴۰۱

فهرست

مقدمه و آشنایی با پروژه

در آخرین فصل تدریس شده با الگوریتم‌های تکاملی آشنا شدیم. هدف این پروژه به کاربردن الگوریتم تکاملی برای یادگیری شبکه‌ی عصبی در محیطی است که داده کافی جهت آموزش آن موجود نیست. یکی از این محیط‌ها بازی است که همواره اتفاق جدیدی در حال رخ دادن است و لذا تولید داده‌های آموزشی جهت آموزش امری نشدنی است.



شکل ۱- نمای اجرای بازی

بازی طراحی شده این پروژه Snail Jumper نامیده شده‌است. این بازی در دو حالت دستی[1] و تکامل عصبی[2] قابل اجرا است. جهت آشنایی با نحوه کار آن، پروژه را از گیت‌هاب دانلود کنید و فایل game.py را اجرا نمایید. پس از اجرا، تصویری مشابه با شکل ۱ تصویر راست مشخص خواهد شد که با انتخاب گزینه اول به صورت دستی

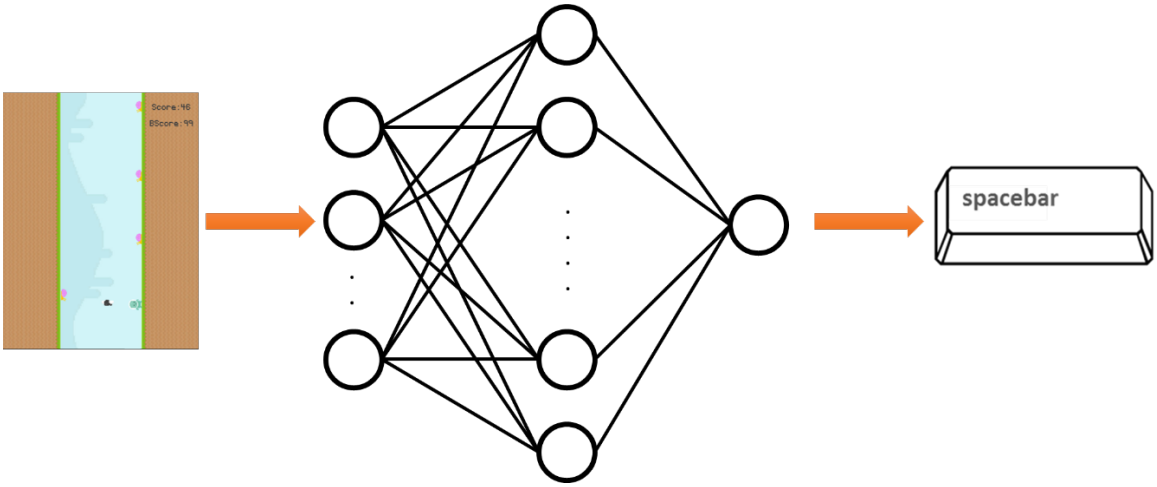
[1] Manual
[2] Neuroevolution

و با انتخاب گزینه دوم به صورت تکامل عصبی می‌توانید آن را اجرا کنید. برای آشنایی، گزینه اول را انتخاب کرده تا بازی اجرا شود. هدف این بازی ساده عبور از موانع سر راه است که با space عمل پرش صورت می‌گیرد.

در ادامه، با نحوه پیاده سازی تکامل عصبی آشنا خواهیم شد و خواهیم دید چگونه الگوریتم تکاملی به یادگیری شبکه عصبی کمک شایانی را خواهند داشت.

شرح مسئله

برای به پیش بردن بازی طراحی شده به صورت تکامل عصبی، باید شبکه عصبی‌ای را طراحی کنیم که **پارامترهای مهم در تصمیم گیری** را تحت ورودی در اختیار بگیرد و سپس خروجی متناظر را تولید کند. در انتها خروجی تولید شده به مشابه فشردن دکمه space تعریف شده در بازی عمل کند.



شکل ۲ – نقش شبکه‌عصبی در الگوریتم تکامل عصبی

بنابراین، پس از تعیین پارامترهای مهم در تصمیم‌گیری و ساختن معماری شبکه‌ی عصبی، عمل feedforward به‌راحتی صورت می‌گیرد. در فصل شبکه‌عصبی آشنا شدیم که در ادامه کار باید تابع هزینه تعریف شود و بعد با backpropagation، وزن‌ها و بایاس‌ها را به گونه ای آپدیت کنیم که به سمت مینیمم میل کند.

اما در مسئله موجود، داده‌ای جهت آموزش و backpropagation وجود ندارد. لذا در این قسمت از الگوریتم‌های تکاملی می‌توان کمک گرفت. بدین صورت که به تعداد زیاد (در پروژه ما ۳۰۰) بازیکن تولید خواهد شد. هر بازیکن حاوی یک شبکه‌ی عصبی است که وزن‌ها و بایاس هرکدام به ترتیب رندم نرمال و صفر مقداردهی اولیه شده‌اند. سپس هریک با توجه به معماری شبکه عصبی و مقادیر اولیه موجود، با مشاهده موانع عملکرد متفاوتی را از خود نشان می‌دهد. تعدادی به موانع برخورد می‌کنند و تعدادی نیز عبور خواهند کرد. هرچه بازیکن به مسیر خود بیشتر ادامه دهد، مقدار شایستگی[3] بیشتری را اختیار خواهد کرد. بنابراین طبق اصل تکامل همواره بازیکن‌های با

[3] fitness

عملکرد بهتر به نسل بعد منتقل خواهند شد و با در نظر گرفتن عملگرهای تقاطع[4] و جهش[5] و با گذر چند نسل، انتظار می‌رود تا عملکرد بهتری را از خود نشان دهند و مسیر بیشتری را طی کنند.

ساختار پروژه

کد پروژه شامل هفت فایل است:

- **game.py**: پیاده‌سازی روند اجرای بازی.
- **evolution.py**: حاوی یک کلاس به نام Evolution برای تکامل موجودات هر نسل.
- **nn.py**: معماری شبکه عصبی و بخش feedforward.
- **player.py**: حاوی کلاس Player برای ساخت بازیکن(ها)ی موجود در صحنه.
- **variables.py**: حاوی متغیرهای عمومی که بین فایل‌ها به اشتراک گذاشته شده‌اند.

[4] crossover
[5] mutation

موارد پروژه

۱) پیاده‌سازی شبکه عصبی (فایل **nn.py**):

در **__init__** کلاس، یک لیست پایتون حاوی تعداد نورون هر لایه دریافت خواهد شد. به عنوان مثال استفاده از [3, 10, 2]، این معنی را می‌دهد که ۳ نورون در لایه ورودی، ۱۰ نورون در لایه پنهان و ۲ نورون در لایه خروجی مورد استفاده قرار خواهد گرفت. شما در این قسمت باید با توجه به ورودی‌های دریافت شده، ماتریس وزن و بردار بایاس های متناظر را ایجاد کنید.

در تابع **activation**، باید یک تابع فعالیت نظیر سیگموید را پیاده سازی کنید.

در تابع **forward** نیز ورودی شبکه عصبی را تحت ورودی تابع در اختیار میگیرید و feedforward را انجام می‌دهید و نورون(ها)ی لایه آخر را در خروجی برمی‌گردانید.

۲) پیاده‌سازی پارامتر های مهم در تصمیم‌گیری مسئله و انتخاب معماری شبکه عصبی (**player.py**):

در **__init__** کلاس و در خط ۳۸، باید معماری مورد استفاده در مسئله را انتخاب کنید. برای مثال لیستی قرار گرفته است که **پاسخ بهینه مساله نیست**. لذا باید با آزمون و خطا به معماری مناسب‌تری دست یابید. توجه شود که یک پاسخ بهینه برای مسئله وجود ندارد و معماری‌های متفاوتی می‌توانند پاسخ بهینه ای را کسب کنند.

در ادامه به پیاده‌سازی تابع **think** بپردازید. به کمک این تابع ابتدا با توجه به ورودی های دریافتی تابع، بردار ورودی شبکه عصبی را تشکیل دهید (پیشنهاد می‌شود این عمل در تابع دیگری انجام دهید و در ابتدای این تابع فراخوانی کنید). توجه کنید که تابع **think** مدام در حین اجرای بازی فراخوانی می‌شود و لذا باید پارامترهایی در تصمیم‌گیری انتخاب گردند که روی انتخاب پرش به سمت چپ یا راست تاثیرگذار هستند. پس از تشکیل بردار ورودی، به کمک **self.nn.forward** (تابع پیاده سازی شده در مورد قبل)، خروجی شبکه‌ی عصبی را تولید و با توجه به خروجی مورد نظر، تابع **self.change_gravity** را فراخوانی کنید.

توجه: همانطور که در کد توضیح داده شده است، در خط ۵۶ تا ۶۰ به صورت تست و نحوه آشنایی با تابع **change_gravity**، کد موقتی قرار داده شده و پس از پیاده سازی شما باید این بخش حذف شود.

توجه: همانطورکه در درس آشنا شدید، اندازه مقادیر ورودی در عملکرد شبکه عصبی تاثیر مهمی را دارند. به عنوان مثال اگر نورون های ورودی مقادیر بسیار بزرگی را اختیار کنند، با تغییر وزن و بایاس اگر از تابع فعالیت

سیگموید عبور کنند همواره مقادیر ۱ را اختیار خواهند کرد. لذا یادگیری مناسبی صورت نخواهد گرفت. بنابراین حتما ورودی های مساله را در یک بازه ای نظیر ۰ و ۱ قرار دهید.

۳) پیاده‌سازی انتخاب بازماندگان (فایل evolution.py):

در صورت اجرای بازی به حالت تکامل عصبی، ۳۰۰ بازیکن ساخته می‌شوند و همگی با توجه به سناریوهای توضیح شده اجرا می‌شوند. پس از پایان یک نسل (باخت تمامی ۳۰۰ بازیکن)، بخش تکامل به آغاز به کار می‌کند. هر بازیکن به میزان فاصله‌ای که طی می‌کند، مقدار شایستگی آن افزایش می‌یابد. لذا تمامی بازیکن ها یک فیلد با عنوان fitness را اختیار می‌کنند.

در این بخش به پیاده‌سازی تابع **next_population_selection** می‌پردازیم. در ورودی یک لیستی از شی بازیکنان (که هرکدام یک فیلدی تحت عنوان fitness برای شایستگی دارند) و به علاوه num_players دریافت می‌شود. سپس با توجه به تعداد num_players که در ورودی دریافت می‌شود، بازماندگان برگردانده می‌شوند. در ابتدا پیاده‌سازی را به این صورت انجام دهید که بازیکن‌ها به کمک مقدار شایستگی sort شوند و num_players تای اول که بهترین شایستگی را دارند انتخاب گردند. سپس چرخه‌ی رولت[6]، SUS و Q-tournament را امتحان کنید و بررسی کنید که کدامیک با توجه به انتخاب‌های دیگر شما برای پیاده‌سازی بخش‌های دیگر، بهتر عمل می‌کنند.

توجه: با توجه به فراموش کار بودن روش (μ, λ)، از روش $(\mu + \lambda)$ استفاده شده است. درنتیجه از اجتماع بازیکنان نسل به اتمام رسیده و نسل قبل آن تحت ورودی دریافت شده است.

۴) پیاده‌سازی انتخاب والدین و تولید موجودات نسل جدید (فایل evolution.py):

پس از انتخاب بازماندگان، حال باید والدین انتخاب گردند و به کمک آن ها نسل بعدی (فرزندان) به وجود آیند.

در این بخش باید تابع **generate_new_population** پیاده‌سازی شود. بازماندگان به عنوان ورودی دریافت می‌شوند و به عنوان خروجی یک آرایه به اندازه num_players از فرزندان برگردانده می‌شود.

در این تابع در صورتی که نسل اول باشد (بازیکنان نسل قبل موجود نباشد)، یک آرایه ای از بازیکنان به صورت رندوم برگردانده می‌شود. در غیر این صورت باید ابتدا والدین انتخاب و سپس فرزندان تولید شوند. به عنوان یک

[6] Roulette wheel

انتخاب، میتوانید تمامی بازماندگان را والد در نظر بگیرید و برای حالت‌های دیگر با چرخه‌ی رولت یا SUS یا Q تورنومنت (هرکدام که عملکرد بهتری را اختیار کرد)، والدین را انتخاب کنید. سپس دو به دو والدین را انتخاب و فرزندان را با عملیات تقاطع و جهش تولید کنید. در انتها لیست فرزندان را بازگردانید.

توجه: فرزندان باید آبجکت‌هایی متفاوت با والدین باشند. لذا برای تولید فرزندان تابع **clone_player** را فراخوانی کنید تا یک نسخه جدید از بازیکن قبلی با همان شایستگی و پارامترهای شبکه عصبی حاصل گردد.

نکته‌ی بسیار مهم:

اهمیت این پروژه در این نکته نهفته است که شما با آزمون و خطا و دانشی که از نحوه‌ی عملکرد شبکه‌های عصبی و الگوریتم‌های تکاملی دارید، هایپرپارامترها و تمامی انتخاب‌هایی را که در این پروژه پیش رو دارید تنظیم کنید تا به یک نتیجه‌ی مطلوب برسید. به همین دلیل نیز است که در هیچ کدام از بخش‌های پروژه شما را ملزم به به کارگیری یک تابع خاص و یا روش خاص نکرده‌ایم (هر چند که تمامی موارد گفته شده باید پیاده‌سازی و امتحان شوند و تنها انتخاب اینکه کدام را در الگوریتم و مدل نهایی خود قرار دهید با شماست). بنابراین در این پروژه ابتدا درباره‌ی انتخاب‌های موجود، در منابع جست و جو کنید. برای مثال به مطالعه‌ی قوانین سرانگشتی و guidelineهای موجود برای انتخاب و طراحی معماری شبکه‌ی عصبی بپردازید و در کلیه‌ی مراحل از انتخاب activation function گرفته تا روش انتخاب بازماندگان، انتخاب‌های مختلف را امتحان کنید و با guidelineها و تحلیل‌هایی که برای الگوریتم‌های تکاملی یاد گرفته‌اید، جست و جو میان حالت‌های مختلف را به صورت هدایت شده پیش ببرید. به عنوان مثال اگر در جایی انتخابی انجام می‌دهید که تنوع را افزایش می‌دهد، در جای دیگر تعادل را برقرار کنید. یا اگر سرعت همگرایی پایینی دارید یک راه حل می‌تواند این باشد که در q- tournament مقدار q را زیاد کنید. یا مثلا بررسی کنید که اگر مشکل همگرایی زودرس اتفاق می‌افتد احتمالا چه دلیلی داشته است و چگونه آن را رفع کردید. دقت داشته باشید که نیازی به نوشتن گزارش نیست اما در تحویل پروژه، از شما خواسته می‌شود تا روند رسیدن به پاسخ بهینه‌ی خود را توضیح دهید و بیان کنید که با گذر از چه حالت‌هایی به انتخاب‌های نهایی خود رسیده‌اید (یعنی باید حضور ذهن کافی برای تمامی موارد گفته شده در این بخش داشته باشید). بنابراین اکیدا توصیه می‌کنیم مواردی را که امتحان کردید به همراه تحلیلی بسیار کوتاه (به صورتی که خودتان بتوانید از روی آن سریع به خاطر بیاورید) یادداشت کنید یا به صورت کامنت را کد خود بنویسید تا برای تحویل آن‌ها را فراموش نکنید. همچنین نیاز است که خصوصا در رابطه با بخش تکاملی پروژه، توجیه منطقی برای انتخاب یا رد یک حالت داشته باشید.

مورد ۵ (امتیازی): منحنی آموزش[۷]

برای تحلیل بهتر فرایند تکامل، در هر نسل، بیشترین، کمترین و متوسط شایستگی بازیکنان را محاسبه و در نهایت پلات کنید. برای این کار، می‌توانید در تابع **next_population_selection**، این کار را انجام دهید. به طوری که در انتهای هر نسل اطلاعات شایستگی آن نسل در یک فایل ذخیره کنید. سپس یک فایل پایتون جدید تعریف کرده و اطلاعات مورد نظر را بخوانید و پلات کنید.

[7] Learning Curve

نحوه تحویل و پل ارتباطی

پروژه را به صورت فایل فشرده در سامانه کورسز بارگذاری کنید.

برای پروژه نیازی به **نوشتن گزارش نیست**. و بعد از مهلت انجام، **تحویل آنلاین** خواهیم داشت.

سوال‌های خود را از طریق ایمیل درس مطرح بفرمایید.