به نام خدا

دانشگاه تهران

پردیس دانشکده‌های فنی

دانشکده برق و کامپیوتر

درس سیستم‌های هوشمند

پروژه پایانی

گلمهر خسروخاور ۸۱۰۱۹۸۵۰۷

سهیل صالحی ۸۱۰۱۹۸۴۲۲

نیلوفر فریدنی ۸۱۰۱۹۸۴۵۲

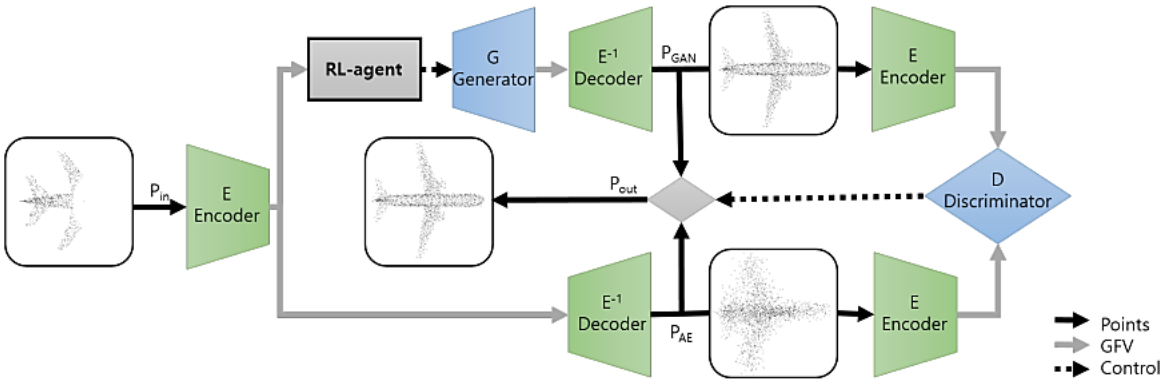
بهمن ۱۴۰۱

فهرست سوالات

آشنایی با RL-GANها

شرح مقاله:

همانطور که در بخش مقدمه شرح داده شد، شبکه‌های GAN قابلیت دارند تا با استفاده از توزیع احتمالی ورودی، خروجی‌ای همانند ورودی تولید کند. اما آموزش چنین شبکه‌هایی سخت هستند و یا هنگام آموزش پایدار نیستند. لذا در مقاله به این نتیجه رسیده شد که قبل از بلوک مولد شبکه GAN، یک عامل یادگیری تعاملی قرار گیرد که بتواند ورودی درستی را به مولد دهد تا مولد به شکل صحیحی، فضای latent ورودی را تولید کند. این عامل یادگیری تعاملی، نیاز به بهینه‌سازی‌های پیچیده را نیز از بین می‌برد.



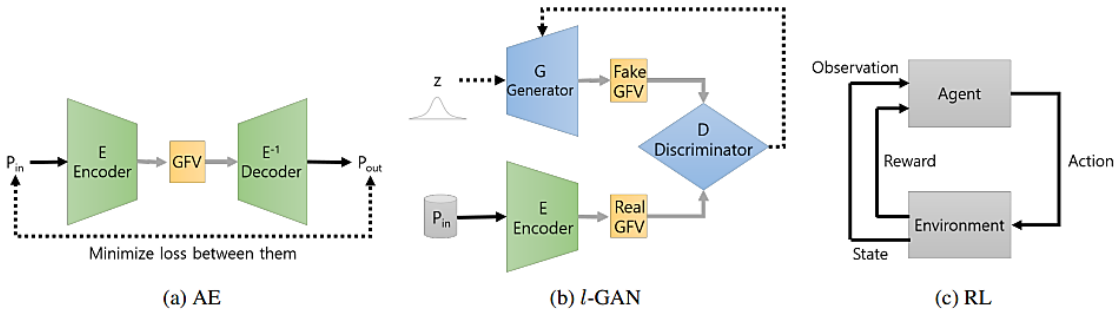
شکل ۱-۱: ساختار شبکه تکمیل‌کننده تصویر

در شکل ۱-۱، شبکه‌ای مشاهده می‌شود که قادر است عکسی که کامل نباشد را تکمیل کند. برای این کار در ابتدا می‌بایست Auto Encoder موجود در شبکه را آموزش داده شود تا فضای latent عکس را بگیرد. سپس عامل شبکه، آموزش می‌بیند تا بهترین بردار z از این فضا به عنوان حرکت انتخاب کند و GANای که از قبل آموزش دیده است بتواند عکسی را با توجه به این بردار تولید کند. بر خلاف شبکه‌های GAN که با استفاده از روش Back-propagation آموزش دیده می‌شدند تا بهترین بردار z را برای تولید بهترین و نزدیک‌ترین عکس به ورودی انتخاب کنند، این شبکه به صورت real-time عمل کرده و همچنین اگر نواحی گم شده، بزرگ‌تر باشد، باز هم مقاوم است. در انتهای شبکه نیز همانند شبکه‌های GAN، یک جدا کننده[۱]، قرار داده می‌شود تا بین خروجی دیکودر Auto Encode و خروجی دیکود شده‌ی شبکه GAN، بهترین را انتخاب و به عنوان خروجی کلی نمایش دهد. قابل ذکر است اگر نواحی گم شده از تصویر کم باشد، یک Auto Encoder تا حد قابل قبولی می‌تواند تصویر درست را نمایش دهد.

اضافه کردن عامل یادگیری تعاملی، باعث حل شدن مشکل ناپایداری شبکه‌های عمیق کنونی می‌شود. همچنین در مقاله ذکر شد که آموزش شبکه‌های GAN بر روی GFV (بردارهای ویژگی global) می‌تواند باعث بهبود پایداری آموزش شود.

---

[۱] Discriminator



شکل ۱-۲: سه بخش اساسی شبکه *RL-GAN-NET*

در شکل ۱-۲ بخش l-GAN را مشاهده می‌کنیم که قسمت مولد، از نویز بر حسب بردار انتخاب شده‌ی z، یک GFV تولید می‌کند و سپس توسط دیکودر Auto Encoder به یک سری نقاط سه بعدی برای خروجی تبدیل می‌شوند. عامل یادگیری تقویتی با توجه به ترکیب تابع خطاهای زیر، بهترین z را برای ورودی GAN انتخاب می‌کند:

- تابع خطای Chamfer که فاصله‌ی Chamfer را بین نقاط Pin ورودی و نقاط تولید شده حساب می‌کند.

$$d_{CH}(P_1, P_2) = \sum_{a \in P_1} \min_{b \in P_2} \|a - b\|_2^2 + \sum_{b \in P_2} \min_{a \in P_1} \|a - b\|_2^2$$

$$L_{CH} = d_{CH}(P_{in}, E^{-1}(G(z)))$$

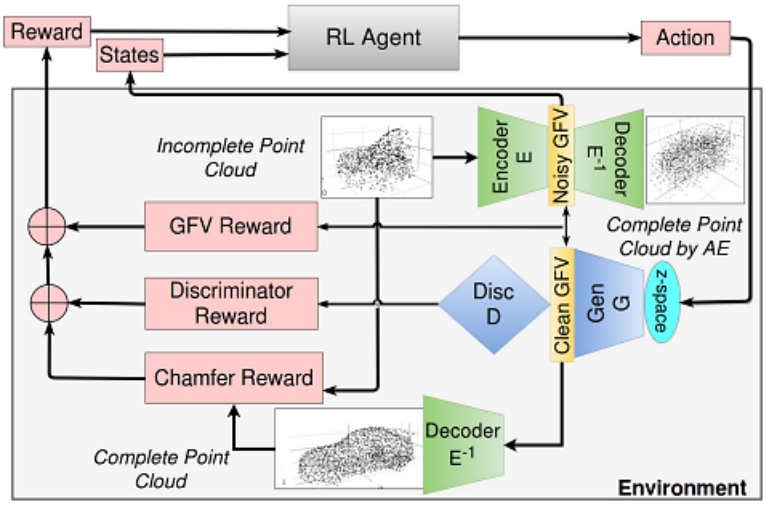
- تابع خطای GFV که نُرم۲ GFV ورودی و GFV zهای منتخب را حساب می‌کند.

$$L_{GFV} = \|G(z) - E(P_{in})\|_2^2$$

- تابع خطای Discriminator که همان خروجی آن است.

$$L_D = -D(G(z))$$

برای عامل یادگیری تعاملی، محیط یا همان environment، ترکیب Auto Encoder و l-GAN است. حالت، همان GFV نویزهای تصویر ناقص است. بهترین تصمیم، بهترین Seed منتخب برای تولید نویز رندوم مولد است.



شکل ۳-۱: آموزش RL-GAN-Net برای تکمیل عکس

به علت آنکه اکشن‌های مختلف برای این ساختار، به صورت پیوسته هستند، از Deep deterministic policy gradient کمک گرفته شده است و آن هم بدین معنی است که یک عامل $\mu(s|\theta^{\mu})$ یک سیاست مشخص را یاد گرفته و حالات را به اکشن‌ها به روشی از پیش تعیین شده map می‌کند. این عامل، به روش امید ریاضی گرفتن از تابع هزینه‌ی زیر، آموزش می‌بینند.

$$\nabla_{\theta^{\mu}} J(\theta) = \mathbb{E}_{s_t \sim \rho^{\beta}} [\nabla_{\alpha} Q(s, a \mid \theta^{Q}) \mid_{s=s_t, a=\mu(s_t)} \nabla_{\theta_{\mu}} \mu(s \mid \theta^{\mu}) \mid_{s=s_t}]$$

در نهایت، به جای بهینه کردن ترکیب توابع خطا، توابع خطا را به پاداش تبدیل می‌شود و سرعت همگرایی و یافتن پاسخ (تکمیل عکس) بسیار بیشتر می‌شود و مدل می‌تواند عکس‌هایی تا حدود ۷۰ درصد داده‌های گمشده را تکمیل کند.

شرح الگوریتم:

Agent Input:

State (s_t): $s_t = GFV_n = \mathbf{E}(P_{in})$; Sample pointcloud P_{in} from dataset into the pre-trained encoder **E** to generate noisy latent representation GFV_n.
Reward (r_t): Calculated using Eq. (5)

Agent Output:

Action (a_t): $a_t = z$
Pass z-vector to the pre-trained generator **G** to form clean latent vector GFV_c=**G**(z)

Final Output:

$P_{out} = \mathbf{E}^{-1}(GFV_c)$; Pass GFV_c into decoder E^{-1} to generate output point cloud P_{out}.

Initialize **procedure Env** with pre-trained generator **G**, discriminator **D**, encoder **E** and decoder $\mathbf{E}^{-1}$
Initialize policy π with **DDPG**, actor **A**, critic **C**, and replay buffer **R**
for $t_{steps} < maxsteps$ **do**
  Get P_{in}
  if $t_{steps} > 0$ **then**
    Train **A** and **C** with **R**
  if $t_{LastEvaluation} > f_{EvalFrequency}$ **then**
    Evaluate π
  $GFV_n \leftarrow \mathbf{E}(P_{in})$
  if $t_{steps} > t_{StartTime}$ **then**
    Random Action a_t
  if $t_{steps} < t_{StartTime}$ **then**
    Use $a_t \leftarrow \mathbf{A} \leftarrow GFV_n$
  $(s_t, a_t, r_t, s_{t+1}) \leftarrow$**Env** $\leftarrow a_t$
  Store transition (s_t, a_t, r_t, s_{t+1}) in **R**
endfor
procedure ENV(P_{in},a_t)
  Get State (s_t) : $GFV_n \leftarrow \mathbf{E}(P_{in})$
  Implement Action : $GFV_c \leftarrow \mathbf{G}\ (a_t = z)$
  Calculate reward r_t using Eq. (5)
  Obtain point cloud : $P_{out} \leftarrow \mathbf{E}^{-1}\ (GFV_c)$

در ابتدا برای محیط، توسط مولد از قبل آموزش داده شده، جدا کننده و انکودر و دیکودر، مقداردهی اولیه را انجام می‌دهیم. سپس، عامل یادگیری تعاملی را که سیاست π را در بر می‌گیرد، توسط روش DDPG مقداردهی اولیه می‌کند. همچنین برای شبکه بازیگر (A)، شبکه منتقد (C) و بافر پاسخ دهنده (R) نیز این کار را انجام می‌دهد.

تعدادی نمونه از نقاط تصویر برمی‌دارد. در یک بازه زمانی مثبت، شبکه‌های بازیگر و منتقد را توسط بافر، آموزش می‌دهد. در بازه زمانی‌ای که بزرگتر از فرکانس EvalFrequency تعریف شده از قبل باشد، سیاست را ارزیابی می‌کند.

توسط انکودر، نقاط نمونه برداری شده از تصویر را، به فضای ویژگی نویزی (GFV) انکود می‌شوند. در این زمان اگر، از زمان شروع گذشته باشیم، عامل، حرکتی را به صورت رندوم انجام می‌دهد (a_t)؛ اما اگر از زمان شروع نگذشته باشیم، شبکه بازیگر، با توجه به فضای ویژگی نویزی (GFV_n) حرکتی را برای عامل

مشخص می‌کند. با دادن اکشن مشخص a_t و با کمک تابع step از محیط Env، حالت، حالت بعدی، پاداش و اطلاعات حرکت مشخص می‌شود. تمامی خروجی‌های step در بافر R ذخیره می‌شوند.

محیط Env اعمال زیر را پیاده سازی می‌کند:

- حالت s_t را با انکود کردن نقاط ورودی به فضای ویژگی نویزی (GFV_n) توسط انکودر از پیش آموزش دیده شده دریافت می‌کند.
- شبکه مولد با تولید فضای ویژگی بدون نویزی (GFV_c) با توجه به اکشن $a_t = z$، اکشنی را پیاده سازی می‌کند.
- طبق معادله $r = w_{CH} \cdot r_{CH} + w_{GFV} \cdot r_{GFV} + w_D \cdot r_D$، پاداش را محاسبه می‌کند.
- توسط دیکودر از پیش آموزش داده شده و با دیکود کردن فضای ویژگی بدون نویز (GFV_c)، نقاط خروجی را بدست می‌آورد.

به زبان ساده، الگوریتم را می‌توان به گونه‌ای شرح داد که گویی شبکه مولد همانند عامل عمل کرده و در محیط، اکشن یا حرکتی انجام می‌دهد و هدف نهایی آن، بیشینه کردن مقدار پاداش است. شبکه جدا کننده نیز سیگنال پاداش را در هر بار به شبکه مولد باز می‌گرداند. بخش‌های انکودر و دیکودر به ترتیب نقاط ورودی را به فضای ویژگی برده و از فضای ویژگی به نقاط خروجی تبدیل می‌کند.

پیاده سازی بازی Connect ۴ با استفاده از Deep Q-learning

مقدمه:

طریقه‌ی طراحی و کارکرد شبکه های عصبی عمیق را در پیشبینی اعمال بهینه شرح دهید. تعیین تابع تلف این شبکه‌ها بر چه مبنایی است؟

در شبکه‌های یادگیری عمیق بر مبنای Q، ورودی همان حالت‌ها و خروجی، مقدار Q پیشبینی شده برای هر اکشن متفاوت است.

طراحی شبکه‌ها عموما به صورت چند لایه تماما متصل است که بنا بر پیچیده یا ساده بودن مسئله، تعداد لایه‌ها و تعداد نورون‌ها در هر لایه می‌تواند تغییر کند. توابع فعال‌ساز غیرخطی tanh و ReLU مورد استفاده قرار می‌گیرند که بخش غیرخطی مسئله را نیز در بر بگیرند.

کارکرد چنین شبکه‌هایی بدین صورت است که اکشن با بیشترین مقدار Q برای حالت کنونی انتخاب می‌شود. شبکه در طی آموزش بر اساس پاداش‌هایی که دریافت می‌کند و همچنین میزان تفاوت مقدار Q پیش بینی شده با مقدار Qای که مشاهده شده است، متناوبا آپدیت می‌شود.

تابع تلف این شبکه‌ها معمولا به صورت Mean squared error بین Qهای پیش‌بینی شده و مقادیر Q مشاهده شده است. لازم به ذکر است مقادیر Q مشاهده شده به صورت جمع پاداش‌های کنونی و دخیل کردن ضریب تخفیف برای پاداش‌های آینده محاسبه می‌شود. این نوع تابع تلف، شبکه را تشویق می‌کند که مقدار Qای را پیش‌بینی کند که پاداش مورد انتظار را برای اکشن به خصوصی، افزایش دهد.

بخش کامپیوتری:

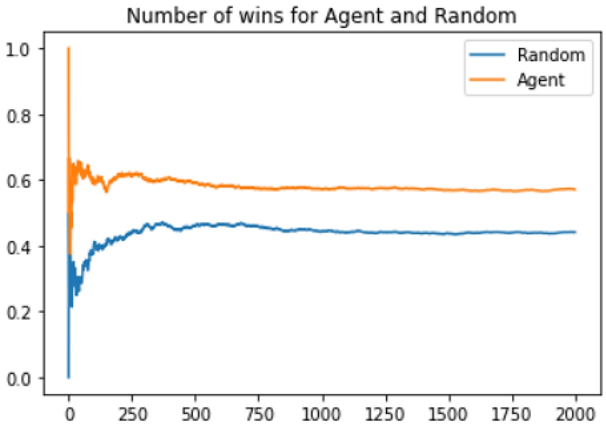
برای پیاده سازی بازی Connect4، از بخش‌های مهم، تعریف کردن مقادیر پاداش برای هر بخش بازی می‌باشد. اگر عامل بازی را برنده شود، پاداش ۵۰ کسب می‌کند؛ اگر ببازد به مقدار ۵۰- جریمه می‌شود؛ اگر بازی بدون آنکه برنده‌ای داشته باشد، تمام شود، عامل مشمول جریمه ۵۰- می‌شود. لازم به ذکر است با قرار دادن مهره درون بازی، عامل پاداشی برابر با ۱/۴۲ دریافت می‌کند(۴۲ خانه بازی). هدف بازی این است که عامل در طی اپیزودهای بسیار و بازی کردن‌های طولانی، بیشترین میزان برد را داشته باشد.

عامل از الگوریتم اپسیلون حریصانه با مقدار اولیه‌ی ۱ استفاده می‌شود و در طی اپیزودها، آن را با ضریب ۰.۹۹۵ کم می‌شود و تا زمانی که به ۰.۰۱ نرسیده، این اعمال را تکرار می‌کند. برای پارامتر یادگیری نیز از مقدار دیفالت پارامتر یادگیری Adam که همان ۰.۰۰۱ است استفاده شد.

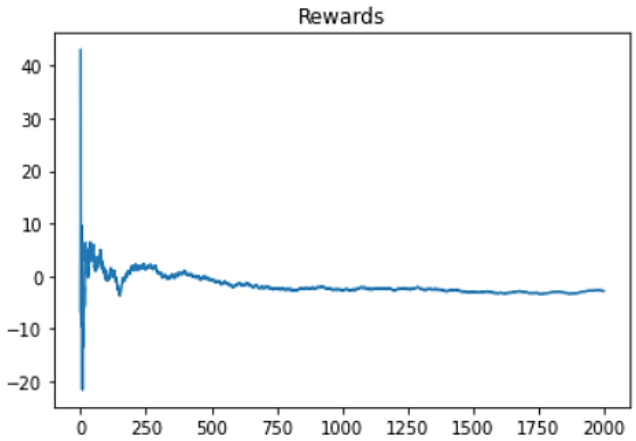
از آنجایی که حریف به صورت رندوم بازی می‌کند، نمی‌توان تنها به پاداش عامل در طول بازی اعتنا کرد. از این رو، تعداد بردهای آن را در مقایسه با بازی به صورت رندوم نیز مقایسه می‌کنیم.

برای آنکه عامل بتواند بهترین عملکرد را داشته باشد، نیاز است تا هر دو طرف نیز خود عامل باشند اما این کار به دلیل حجم زیاد محاسبات و زیاد شدن حالت های بازی، زمان زیادی می‌گیرد. به همین علت، از حریف به صورت رندوم برای آموزش این عامل استفاده می‌شود.

برای بخش شبکه‌ی عصبی آن، در ابتدا از ۳ لایه Dense استفاده می‌کنیم که در هر لایه ۳۲ نورون وجود دارد.

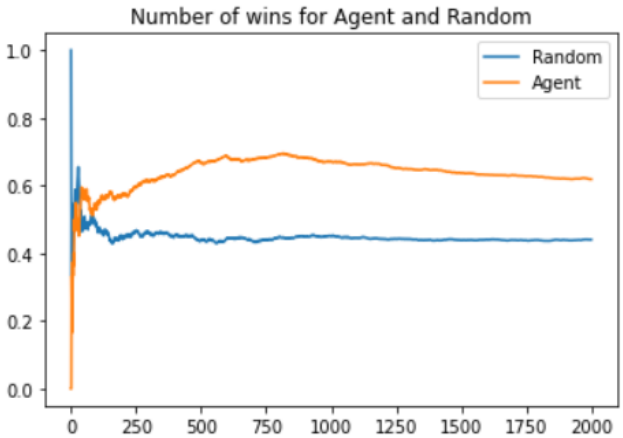


شکل۲-۱: تعداد برد ها (تقسیم بر تعداد کل بازی) برای مدل و به صورت رندوم

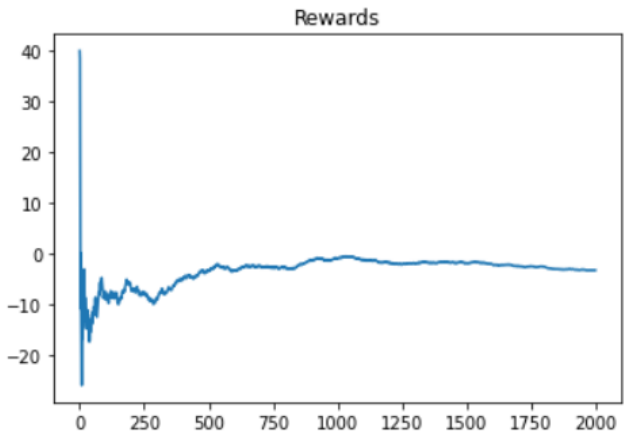


شکل۲-۲ : پاداش دریافتی مدل در طول بازی

سپس تعداد نورون‌های هر لایه را به ۶۴ رسانده و نتایج را مشاهده می‌کنیم:



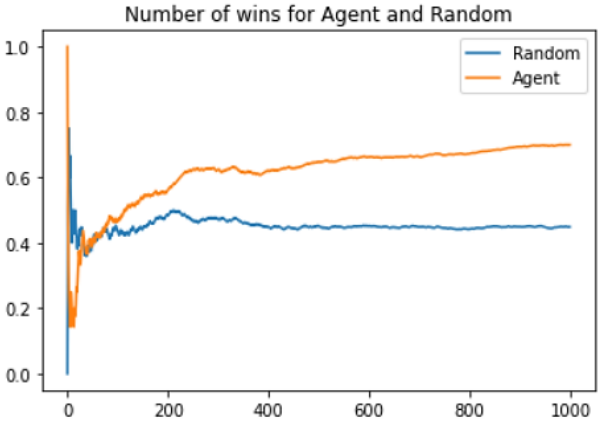
شکل۲-۳: تعداد برد ها (تقسیم بر تعداد کل بازی) برای مدل و به صورت رندوم



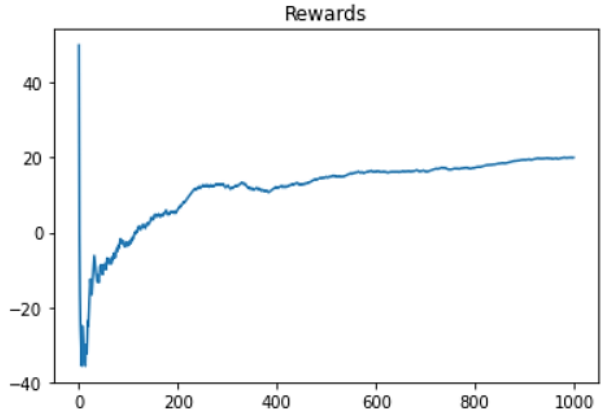
شکل۲-۴: پاداش دریافتی مدل در طول بازی

حال مشاهده می‌کنیم که اگر به لایه‌ها لایه‌ی کانوولوشن اضافه کنیم، نتایج شبکه چه تغییری می‌کند.

با قرار دادن یک لایه Conv دو بعدی با سایز فیلتر ۵*۵ و تعداد ۳۲، نتایج را مشاهده می‌کنیم:

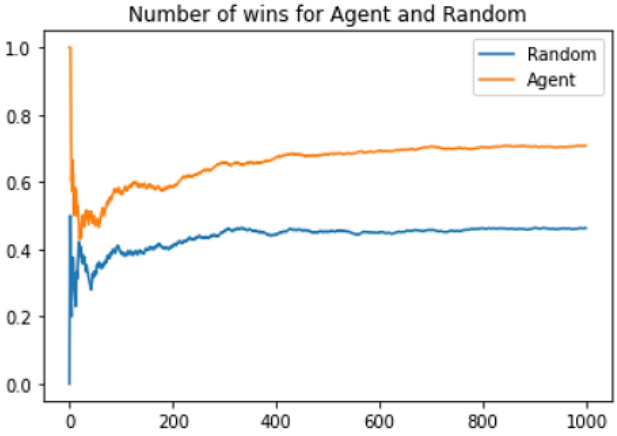


شکل۲-۵: تعداد برد ها (تقسیم بر تعداد کل بازی) برای مدل و به صورت رندوم

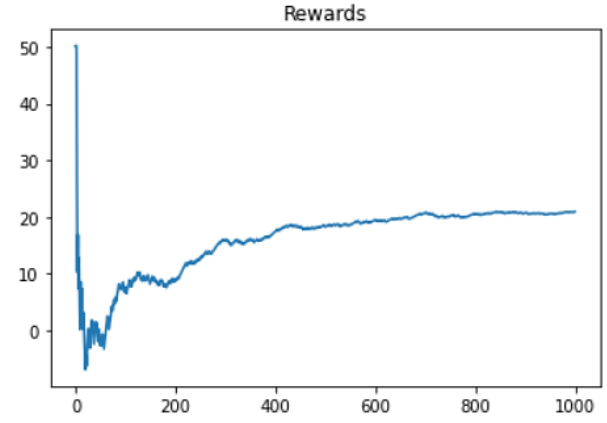


شکل۲-۶: پاداش دریافتی مدل در طول بازی

حال صرفا در لایه‌ی Conv دو بعدی، سایز فیلتر را به ۳*۳ کاهش داده و تعدادشان را به ۶۴ می‌رسانیم:

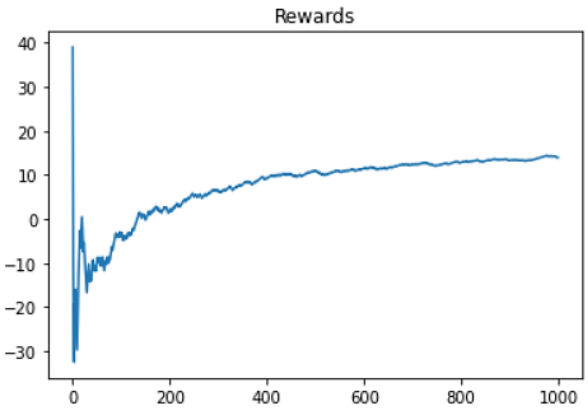


شکل۲- ۷: تعداد برد ها (تقسیم بر تعداد کل بازی) برای مدل و به صورت رندوم

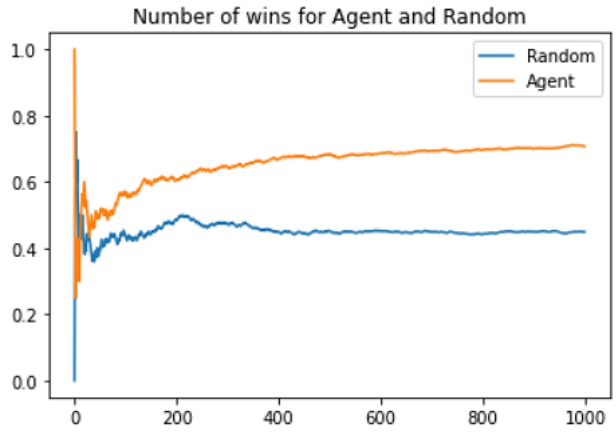


شکل۲-۸: پاداش دریافتی مدل در طول بازی

مشاهده می‌شود که تا کنون شاهد بهترین نتیجه یعنی بهترین تعداد برد در ۱۰۰۰ اپیزود بودیم. برای بهتر شدن نتایج، تصمیم بر تغییر مقدار جریمه به ازای قرار دادن هر مهره در بازی به اندازه ۱- گرفته شد. حال همان نتایج را به ترتیب مشاهده می‌کنیم.



شکل۲-۱۰: پاداش دریافتی مدل در طول بازی



شکل۲-۹: : تعداد برد ها (تقسیم بر تعداد کل بازی) برای مدل و به صورت رندوم

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 6, 7, 64)          1664

 conv2d_1 (Conv2D)           (None, 6, 7, 64)          16448

 flatten (Flatten)           (None, 2688)              0

 dense (Dense)               (None, 64)                172096

 dense_1 (Dense)             (None, 128)               8320

 dense_2 (Dense)             (None, 64)                8256

 dense_3 (Dense)             (None, 7)                 455

=================================================================
Total params: 207,239
Trainable params: 207,239
Non-trainable params: 0
_________________________________________________________________
```

شکل ۲-۱۱: معماری بخش عمیق شبکه

در شکل‌های ۲-۹ و ۲-۱۰ مشاهده می‌شود که بهترین نتایج پیاده سازی بازی Connect4 به ازای قرار دادن یک لایه کانوولوشنی ۲ بعدی با سایز فیلتر ۳*۳ و تعداد ۶۴، سپس ۳ لایه Dense با تعداد نورون ۶۴ در ساختار شبکه است. اگر عامل بازی را برنده شود، پاداش ۵۰ کسب می‌کند؛ اگر ببازد به مقدار ۵۰- جریمه می‌شود؛ اگر بازی بدون آنکه برنده‌ای داشته باشد، تمام شود، عامل مشمول جریمه ۵۰- می‌شود همچنین جریمه حرکت ۱- است.

پیوست:

https://www.voigtstefan.me/post/connectx/

https://www.kaggle.com/code/phunghieu/connectx-with-q-learning/notebook

https://keon.github.io/deep-q-learning/

https://medium.com/@louisdhulst/training-a-deep-q-learning-network-for-connect-4-9694e56cb806

https://www.kaggle.com/code/alexisbcook/deep-reinforcement-learning

https://deeplizard.com/learn/video/mo96Nqlo1L8