به نام خدا

دانشگاه تهران

دانشکده مهندسی برق و کامپیوتر

پروژه نهایی

سیستم‌های هوشمند

دکتـــر حســـینی

امیرحسین بیرژندی ۸۱۰۱۹۸۳۶۷

عرفـان پـنـــــاهی ۸۱۰۱۹۸۳۶۹

نیمسال اول ۰۲-۱۴۰۱

فهرست:

***** فایل شبیه سازی بخش دوم در محیط پایتون با نام IS_ProjectB_Q2_810198367_810198369.ipynb پیوست شده است.**

***** فایل ارائه پروژه با نام IS_ProjectB_Presentation_810198367_810198369.ipynb پیوست شده است.**

چکیده : هدف از پروژه

در این پروژه قصد داریم در دو بخش، برخی کاربرد های یادگیری تقویتی (Reinforcement Learning) آشنا شویم. در بخش اول پس از آشنایی با GAN ها می‌خواهیم با ترکیب آن‌ها با یادگیری تقویتی، شبکه های RL-GAN را مورد بررسی قرار دهیم و کاربرد آن‌ها در تکمیل تصاویر با نویز ابرشکل را مشاهده کنیم. در این بخش، از مقاله زیر استفاده می‌کنیم و الگوریتم آن‌را توضیح می‌دهیم.

RL-GAN-Net: A Reinforcement Learning Agent Controlled GAN Network for Real-Time Point Cloud Shape Completion (Link)

در بخش دوم، می‌خواهیم به پیاده‌سازی بازی Connect4 با استفاده از Deep Q-Learning بپردازیم. در این بخش ابتدا از محیط (Environment) را با استفاده از کتابخانه kaggle-environment وارد کرده و سپس الگوریتم Deep Q-Learning را برای یادگیری Agent استفاده می‌کنیم. در این بخش از منابع زیر برای فهم و درک بیشتر محیط بازی و پیاده‌سازی الگوریتم استفاده شده است.

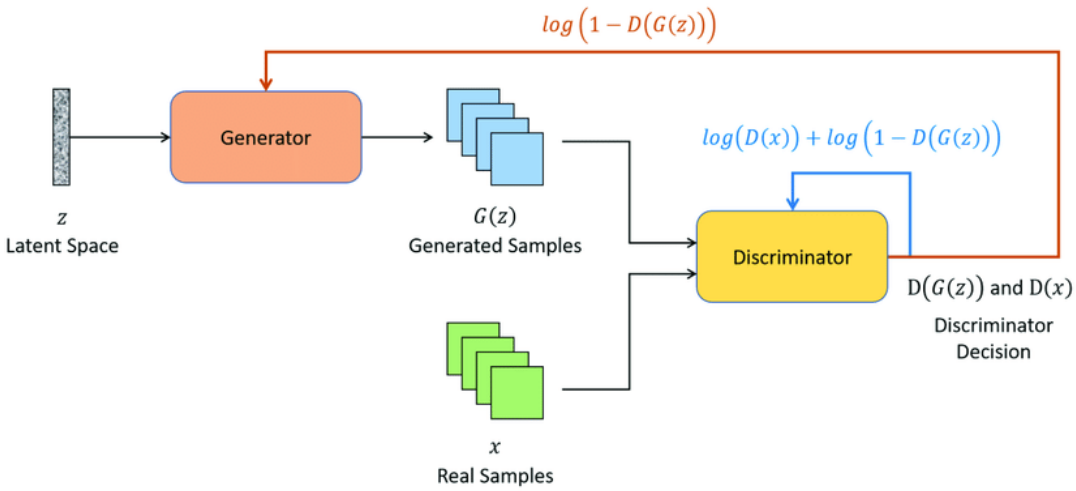
- https://www.kaggle.com/code/ajeffries/connectx-getting-started/notebook
- https://www.kaggle.com/code/gordotron85/teaching-an-agent-to-play-connect-4
- https://medium.com/@louisdhulst/training-a-deep-q-learning-network-for-connect-4-9694e56cb806

بخش اول : آشنایی با RL-GAN ها

در این بخش ابتدا توضیحات مختصری راجع به شبکه های GAN خواهیم داد و سپس در خصوص بخش های مهم مقاله صحبت می‌کنیم و به سوالات انتهایی پاسخ می‌دهیم.

آشنایی با GAN ها: GAN ها دارای دو مدل (شبکه) تولید کننده (Generator) و جداکننده (Discriminator) هستند که هرکدام به‌طور مداوم در تماس با دیگری است. تولید کننده سعی می‌کند با تخمین توزیع احتمال ورودی و یک بردار ورودی نویزی، خروجی را تا حد امکان به داده های اصلی نزدیک کند. در طرف مقابل جداکننده سعی می‌کند داده های واقعی (اصلی) را داده های تولیدی (ساختگی) تفکیک کند. این دو بخش دائماً با هم در تعامل هستند و هم‌دیگر را تقویت می‌کنند. در حقیقت شبکه تولید کننده برای آموزش شبکه جداکننده استفاده می‌شود و برای جداکننده داده تولید می‌کند. در نتیجه این فرآیند نوعی آموزش بدون نظارت (Unsupervised Learning) است. از کاربرد های شبکه جداکننده می‌توان به تشخیص اصلی یا تقلبی بودن اسکناس های پول اشاره کرد.

هر کدام از بخش ها درتضاد با دیگری قصد دارند تابع هدف مسئله را به نفع خود کمینه یا بیشینه کنند. در نتیجه این دو بخش دائماً همدیگر را به پیشرفت وادار می‌کنند تا در نهایت به یک نقطه تعادل (پایداری) برسند. شبکه تولید کننده یک تلف دارد تا اگر خروجی اش به حد کافی به داده های اصلی شبیه نیست جریمه شود و در طرف مقابل جداکننده نیز یک تابع تلف دارد تا درصورتی که نتواند داده واقعی را از ساختگی تشخیص دهد جریمه شود. در نتیجه مطابق با تصویر ۱-۱ تابع هدف GAN به صورت زیر تعریف می‌شود که با یک مسئله کمینه بیشینه (minmax) رو برو هستیم.



تصویر ۱-۱: بلوک دیاگرام GAN

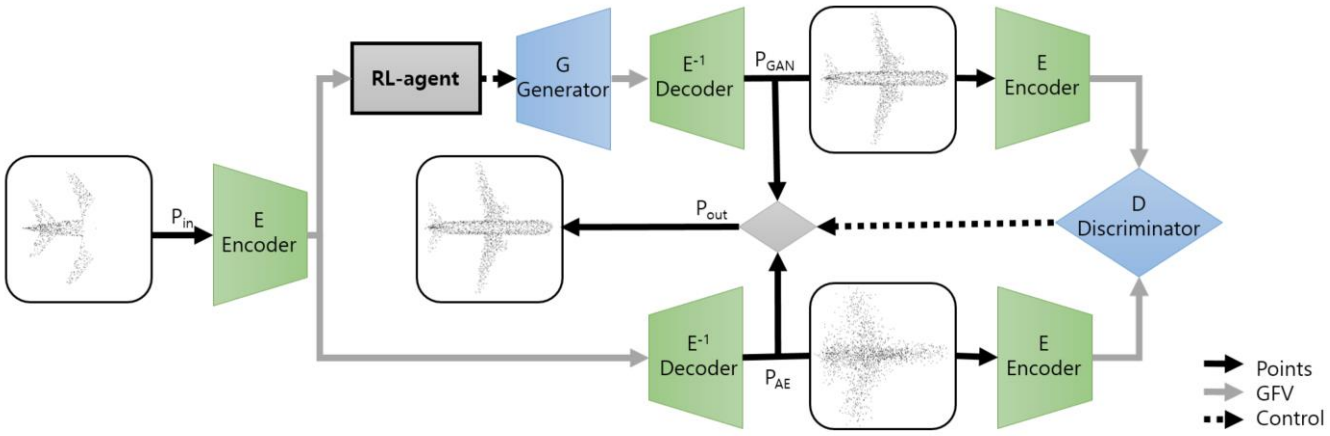
$$\min_G \max_D V(D,G)$$

$$V(D,G) = \mathbb{E}_{x\sim p_{data}(x)}[\log(D(x)] + \mathbb{E}_{z\sim p_Z(z)}\left[\log\left(1 - D\left(G(z)\right)\right)\right]$$

حال می‌خواهیم با الگوریتم و طریقهٔ کارکرد **RL-GAN** ها در مقالهٔ داده شده آشنا شویم.

در خیلی از کاربردها زمانی که با داده سه بعدی سروکار داریم آن ها را به صورت point cloud استفاده می‌کنیم. اکثرا این داده ها زمانی که به صورت point cloud هستند دارای کم و کاستی هایی هستند و ممکن است بخشی از آن ها ناقص باشند. در این مقاله نیز با استفاده از ترکیب یادگیری تقویتی و GAN ها متدی طراحی شده که این نواحی خالی را به بهترین نحو پر می‌کند. در واقع در این متد عامل یادگیری است که با یک شبکه GAN به نوعی کنترل می‌کند که بتواند داده های ناقص را پیش‌بینی کند و شکل کامل آن ها را بسازد.

در حقیقت تأثیر RL-Agent در معماری GAN این است که نویز ورودی شبکه Generator را به‌صورت تصادفی تولید نمی‌کند و در شبکه RL-GAN به‌طور هوشمند تولید می‌شود. تصویر ۱-۲ معماری شبکه RL-Agent برای تکمیل ورودی (پر کردن نقاط) را نشان می‌دهد. (آموزش شبکه های GAN روی GFV که بردار های ویژگی است باعث بهبود فرایند یادگیری می‌شود.)



تصویر ۱-۲: معماری شبکه RL-GAN

هسته اصلی تشکیل دهنده متد RL-GAN از سه بخش تشکیل شده است.

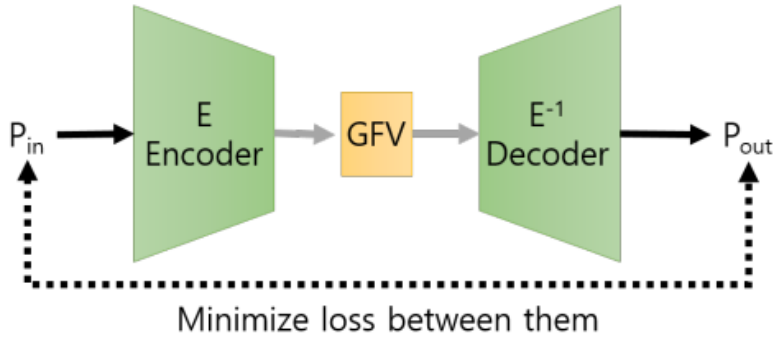
۱- Auto Encoder

۲- latent space GAN

۳- RL Agent

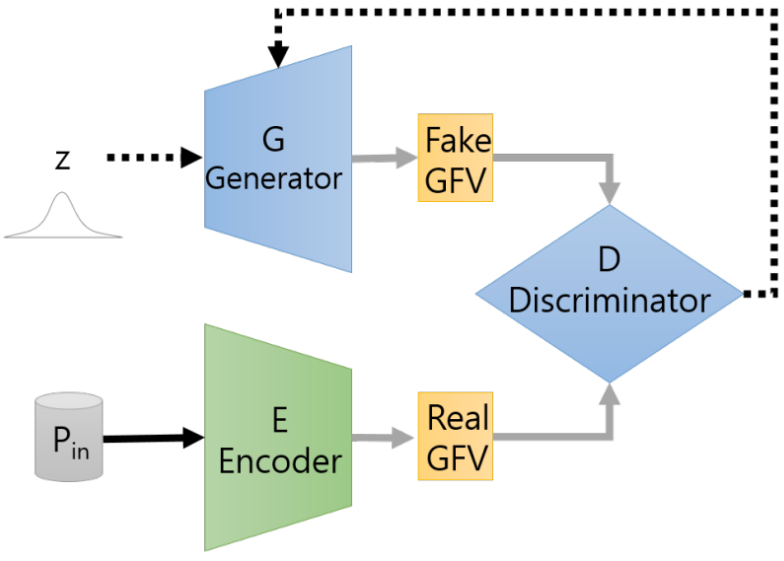
هر کدام از این سه بخش در واقع یک شبکه عصبی هستند که نیاز به آموزش جداگانه دارند. ابتدا AE را آموزش می‌دهیم و با خروجی های انکود شده AE شبکه های l-GAN را ترین می‌کنیم و عامل یادگیری نیز با تعامل با این دو شبکه آموزش دیده خواهد شد.

تصویر ۱-۳، بخش Auto Encoder را نشان می‌دهد. همانطور که در تصویر ۱-۲ نیز مشاهده می‌شود، هدف این است که این خطای بین خروجی و ورودی در این بخش کمینه شود.



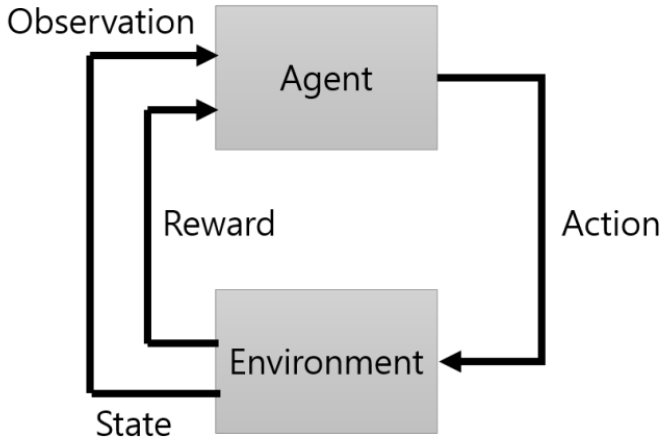
تصویر ۱-۳: معماری بخش Auto Encoder

تصویر ۱-۴، بخش latent space GAN را نشان می‌دهد. همانطور که در توضیحات ابتدایی نیز اشاره شد از این بخش که ساختار یک شبکه GAN را نمایش می‌دهد برای یادگیری دو شبکه Generator و Discriminator استفاده شده است.



تصویر ۱-۴: معماری بخش latent space GAN

در ساختار بالا ورودی z شبکه Generator باید با استفاده از RL-Agent تعیین شود که ساختار این بخش نیز در تصویر ۱-۵، داده شده است.



تصویر ۱-۵: معماری بخش RL

در فرایند یادگیری سه تابع تلف برای سه بخش Discriminator، Auto Encoder و خروجی Generator در نظر گرفته شده است. ابتدا معیار Chamfer Distance را به‌صورت زیر تعریف می‌کنیم.

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

حال تابع تلف هر بخش را به‌صورت زیر تعریف می‌کنیم.

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

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

$Discriminator\ Loss$: $L_{CH} = -D(G(z))$

حال با استفاده از توابع تلف بالا، Reward بخش RL را نیز به صورت زیر تعریف می‌کنیم. (ω ها وزن‌های هر تابع تلف هستند.)

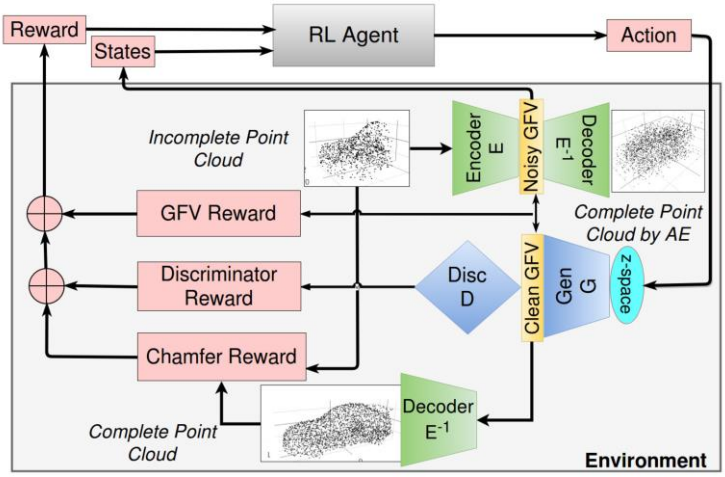
$r_{CH} = -L_{CH}$

$r_{GFV} = -L_{GFV}$

$r_D = -L_D$

$r = \omega_{CH}.r_{CH} + \omega_{GFV}.r_{GFV} + \omega_D.r_D$

در نهایت آموزش با استفاده از RL-GAN-net را مطابق با تصویر ۱-۶ مدل می‌کنیم. همانطور که در تصویر مشاهده می‌شود Reward کل با استفاده از پاداش خروجی هر شبکه به‌دست می‌آید. همچنین حالت (state) ورودی RL-Agent با استفاده از ترکیب حالت خروجی (در حقیقت خروجی Generator) و خروجی Auto Encoder بدست می‌آید.



تصویر ۱-۶: ساختار مدل RL-GAN-net

الگوریتم مربوط به پیاده‌سازی RL-GAN نیز مطابق با تصویر ۱-۷ خواهد بود.

Algorithm 1 Training RL-GAN-Net

Agent Input:

State (s_t): $s_t = GFV_n$ = **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
  $GFV_n \leftarrow$ **E**(P_{in})
  if $t_{steps} > t_{StartTime}$ **then**
    Random Action a_t
  if $t_{steps} < t_{StartTime}$ **then**
    Use $a_t \leftarrow$ **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) : $GFV_n \leftarrow$ **E**(P_{in})
  Implement Action : $GFV_c \leftarrow$ **G** $(a_t = z)$
  Calculate reward r_t using Eq. (5)
  Obtain point cloud : $P_{out} \leftarrow \mathbf{E}^{-1}$ (GFV_c)

تصویر ۱-۷: الگوریتم RL-GAN-net

همانطور که در تصویر ۱-۷ نیز مشاهده می‌شود، ورودی های RL-Agent، پاداش کل (Reward) و حالت (state) است. در خصوص پاداش کل صحبت کردیم. ورودی حالت نیز، همان تصویر اولیه (P_{in}) انکود شده است. همچنین خروجی RL-Agent نیز بردار ورودی به شبکه Generator است. برای بدست آوردن policy و همچنین Q-value ها در هر مرحله از الگوریتم DDPG (Deep Deterministic Policy Gradient) استفاده می‌کنیم. در این الگوریتم دو شبکه actor و critic وجود دارد که شبکه critic با استفاده از معادله Bellman به‌روز می‌شود و شبکه actor با استفاده از گرادیان تابع هزینه که به‌صورت زیر است جریمه (به‌روز رسانی) می‌شود. (به عبارت زیر policy gradient نیز گفته می‌شود.)

$$\nabla_{\theta^\mu} J(\theta) = \mathbb{E}_{s_t \sim \rho^\beta} \left[\underbrace{\nabla_\alpha Q(s, a|\theta^Q)}_{critic\ network} \underbrace{\nabla_{\theta^\mu} \mu(s|\theta^\mu)}_{actor\ network} \right]_{s=s_t, a=\mu(s_t)}$$

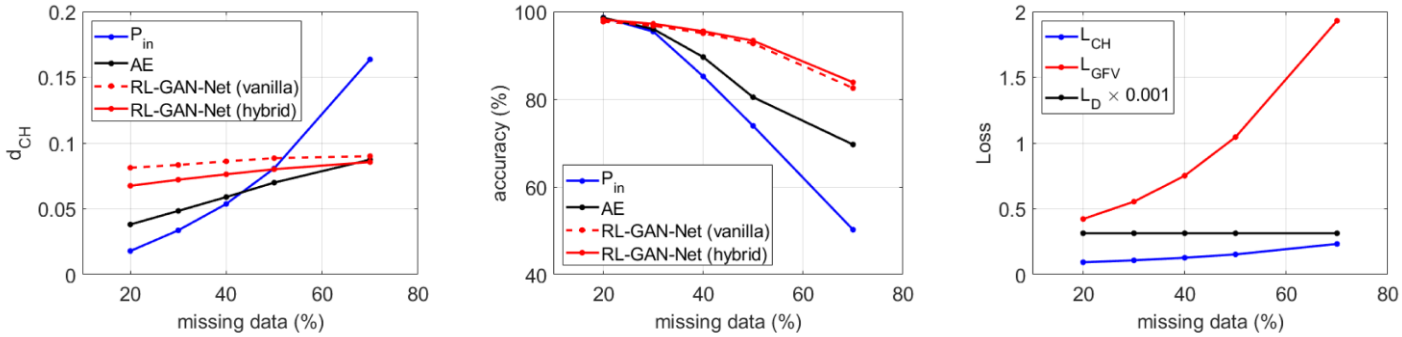
نکته قابل توجه در مورد معماری شبکه این است که شبکه های Generator و Discriminator از پیش آموزش داده شده اند. خروجی Auto Encoder نیز با P_{out} نشان داده می‌شود. در فرایند یادگیری (مشابه با Deep Q-learning) شبکه‌ها با استفاده از حافظه‌ای ذخیره شده برای حرکات (actions)، حالات (states) و پاداش‌ها (rewards) به‌روز می‌شود. فرآیند به‌روزرسانی در هر بار دریافت ورودی P_{in} یک‌بار اتفاق می‌افتد. به‌طوریکه پس از دریافت ورودی P_{in} در ابتدا RL-Agent یک حرکت تصادفی و در ادامه با استفاده از فضای ویژگی نویزی GFV_n دریافت کرده، حرکت (a_t) را انتخاب می‌کند. در نهایت نیز دیکودر با دیکود کردن فضای ویژگی بدون نویز GFV_c، تصویر پر شده (نقاط اضافه شده) را خروجی می‌دهد. تمامی این قسمت ها در تصویر ۱-۲ نیز قابل مشاهده است.

با شرح الگوریتم بالا می‌توانیم کارکرد کلی RL-GAN-net را به طور ساده تربه این صورت توصیف کنیم:

«شبکه Generator در حقیقت حکم Agent را در یک مسئله RL دارد که سعی می‌کند پاداش دریافتی را بیشینه کند. همچنین شبکه Discriminator، در هر مرحله پاداش را به Agent باز می‌گرداند.»

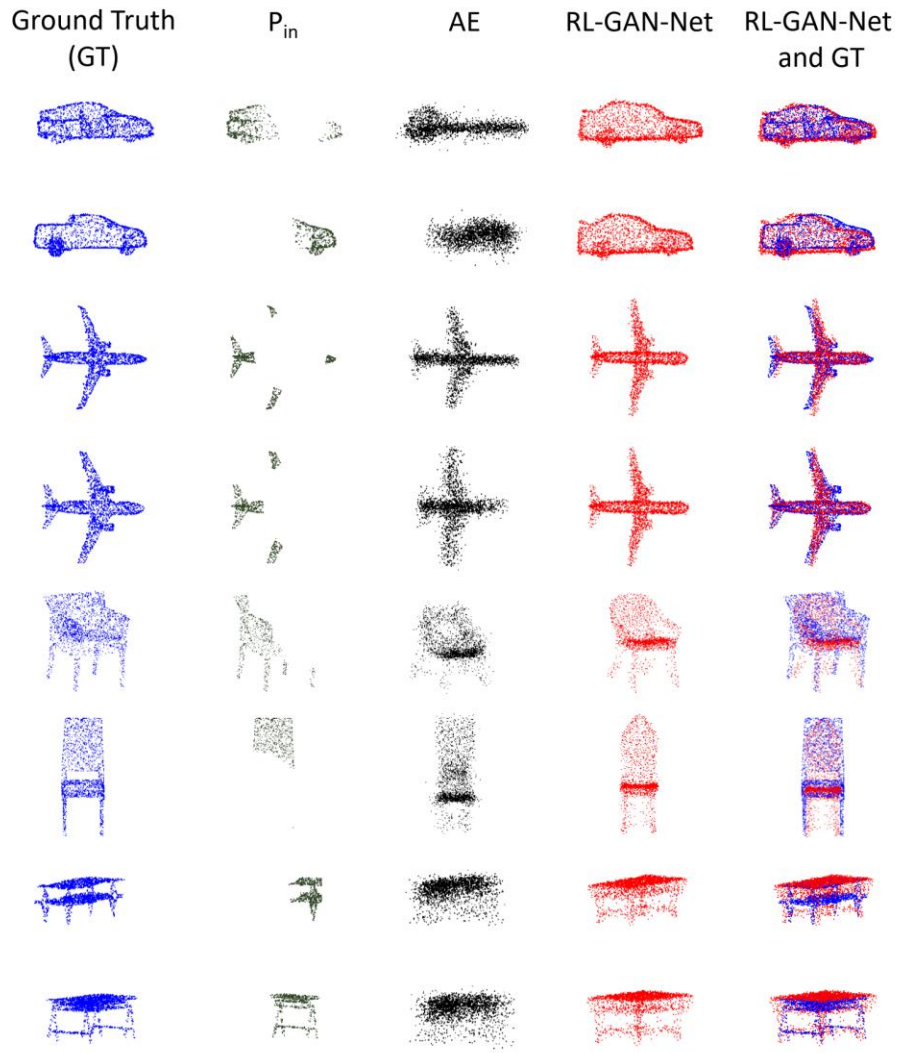
ارزیابی کارایی مدل: تصویر ۱-۸ مقایسه پارامتر های مدل برای درصد نقاط از بین رفته متفاوت را نشان می‌دهد. همانطور که مشاهده می‌شود، با کاهش نقاط (افزایش درصد نقاط از بین رفته) تصویر اولیه، دقت مدل کاهش پیدا می‌کند.

همچنین تصویر ۱-۹ نیز مقایسه ای از خروجی مدل را برای حالتی که ۷۰٪ نقاط ورودی از بین رفته است را نشان می‌دهد.



(a) Chamfer distance to GT (b) Classification accuracy [34] (c) Loss terms

تصویر ۱-۸: ارزیابی کارایی مدل

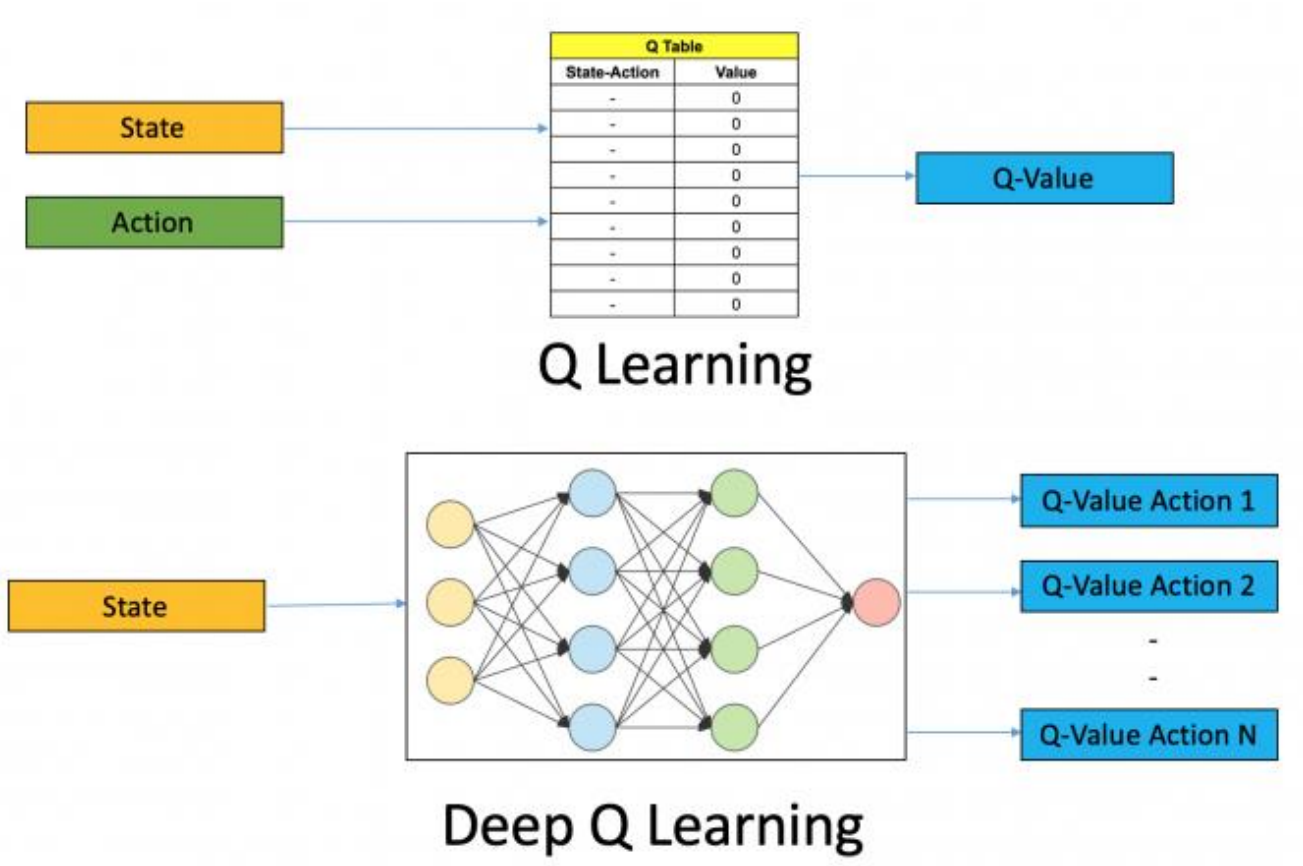


تصویر ۱-۹: نتایج و خروجی مدل (برای حالتی که ۷۰٪ نقاط ورودی از بین رفته است)

بخش دوم : پیاده‌سازی بازی connect4 با استفاده از Deep Q-Learning

در این بخش می‌خواهیم با مبحث Deep Q-Learning آشنا شویم و از این روش برای پیاده‌سازی بازی connect 4 استفاده کنیم. به این منظور در ابتدا توضیحات مختصری راجع به Deep Q-Learning داده و شبکه‌های عصبی مورد استفاده در آن می‌دهیم. سپس در خصوص تابع تلف مورد نظر برای شبکه مورد نیاز توضیح می‌دهیم.

همانطور که در صورت پروژه نیز توضیح داده شده است، زمانی که تعداد state های بازی خیلی زیاد باشد، استفاده از Q-table به حافظه زیادی احتیاج دارد. در نتیجه باید به دنبال راهکار دیگری باشیم. برای مثال بازی connect4 شامل حدودا ۴.۵ هزار میلیارد حالت مختلف است. در نتیجه امکان ذخیره سازی بهترین اکشن انتخابی در هر حالت را نداریم. راهکار مناسب‌تر برای انتخاب بهترین حرکت در هر حالت، یادگیری یک شبکه عصبی است که حالت را به‌عنوان ورودی دریافت کرده و Q-value هر حرکت را به‌عنوان خروجی بدهد. در نهایت با تشخیص بزرگترین Q-value، بهترین حرکت ممکن را انجام می‌دهیم. تصویر ۲-۱ مقایسه‌ای از دو روش Q-Learning و Deep Q-Learning را نشان می‌دهد.



تصویر ۲-۱: مقایسه روش Q-Learning و Deep Q-Learning

- طریقهٔ طراحی و کارکرد شبکه‌های عصبی عمیق را در پیش‌بینی اعمال بهینه شرح دهید.

همانطور که گفته شد از شبکه عصبی برای بدست آوردن بهترین حرکت ممکن در هر حالت استفاده می‌کنیم. یعنی حالت را به شبکه می‌دهیم و حرکت را از آن می‌گیریم. وزن‌های شبکه پس از پایان هر دور بازی فارغ از پیروزی یا شکست، با استفاده از اطلاعات ذخیره شده به‌روز می‌شود. این اطلاعات شامل موارد زیر است:

۱- صفحهٔ بازی پیش از انجام حرکت

۲- حرکت انتخاب شده

۳- پاداش در ازای انجام حرکت

این اطلاعات را در هر حرکت از بازی ذخیره می‌کنیم و پس از پایان دور بازی با آن شبکه را به‌روز رسانی می‌کنیم. به این منظور یک class تعریف می‌کنیم و پس از هر حرکت اطلاعات آن را اضافه می‌کنیم و در نهایت پس از پایان دور و به‌روز شدن شبکه، آن را خالی (clear) می‌کنیم.

برای به‌روز رسانی، از تابع هزینه Sparse Cross Entropy استفاده می‌کنیم. به این منظور از خروجی ها شبکه و پاداش دریافتی کل استفاده می‌کنیم تا مقدار تابع هزینه را بدست آوریم. سپس در مرحله Back Propagation گرادیان وزن های هر لایه را بدست می‌آوریم و در نهایت با استفاده از بهینه ساز Adam، مقادیر وزن‌ها را به‌روز رسانی (بهینه) می‌کنیم.

- تعیین تابع تلف این شبکه ها بر چه مبنایی است؟

تفاوت به‌روز رسانی شبکه در Deep Q-Learning و تابع هزینه این شبکه این است که باید پاداش هر حرکت نیز در محاسبه خطا تأثیر داشته باشد. با استفاده از Q-value های حرکات ممکن برای صفحهٔ حال حاضر بازی و همچنین Q-value حرکت بعدی هزینه (خطا) را بدست می‌آوریم. با استفاده از رابطهٔ به‌روز رسانی Q-value ها، خطا را با استفاده از دو پارامتر زیر تعریف می‌کنیم.

$$Q_{new}(s_t, a_t) = Q_{old}(s_t, a_t) + \alpha \left(\overbrace{\underbrace{R_t + \gamma \max Q(s_{t+1}, a_{t+1})}_{Target} - \underbrace{Q_{old}(s_t, a_t)}_{Prediction}}^{TD\ error} \right)$$

$$TD\ error = R_t + \gamma \max Q(s_{t+1}, a_{t+1}) - Q_{old}(s_t, a_t)$$

در نیتجه تابع هزینه با استفاده از دو پارامتر $Target$ و $Prediction$ بدست می‌آید.

با استفاده از توضیحات بالا، شبکه عصبی را برای یادگیری تقویتی پیاده‌سازی می‌کنیم. حال می‌خواهیم نتایج این پیاده‌سازی را بررسی کنیم.

برای مشاهده روند یادگیری شبکه، دو نمودار را رسم می‌کنیم.

نمودار تعداد پیروزی: در این نمودار تعداد پیروزی ها تا هر اپیزود را نشان می‌دهد.

نمودار مجموع پاداش دریافتی: در این نمودار مجموع پاداش دریافتی تا هر اپیزود رسم می‌شود. (برای بهتر شدن شکل نمودار از دستور ()smoothing استفاده شده است.)

سیاست پاداش یا جریمه: در این بازی پاداش پیروزی $50+$، جریمه هر شکست $50-$ و پاداش هر حرکت را 1 یا $1-$ در نظر می‌گیریم. برای تعیین پاداش یا جریمه هر حرکت دو استراتژی داریم:

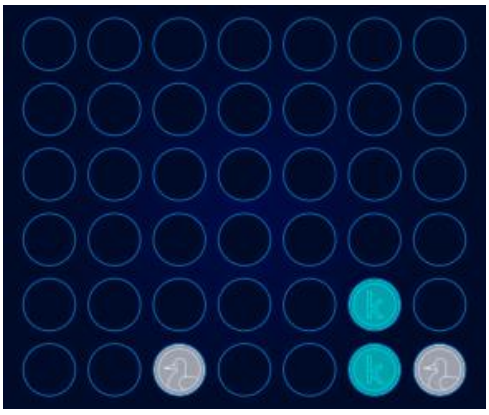
۱- اگر پاداش هر حرکت را $1+$ در نظر بگیریم بازی طولانی تر می‌شود و تعداد حالت های بازی که شبکه با آن آموزش داده می‌شود بیشتر و بیشتر می‌شود.

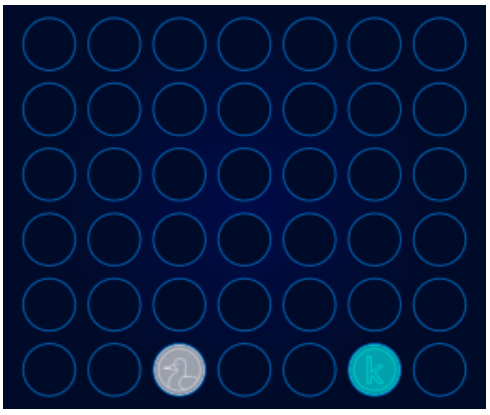
۲- اگر پاداش (جریمه) هر حرکت را $1-$ در نظر بگیریم در فرآیند یادگیری این موضوع را لحاظ می‌کنیم که Agent سریع‌تر بازی را ببرد.

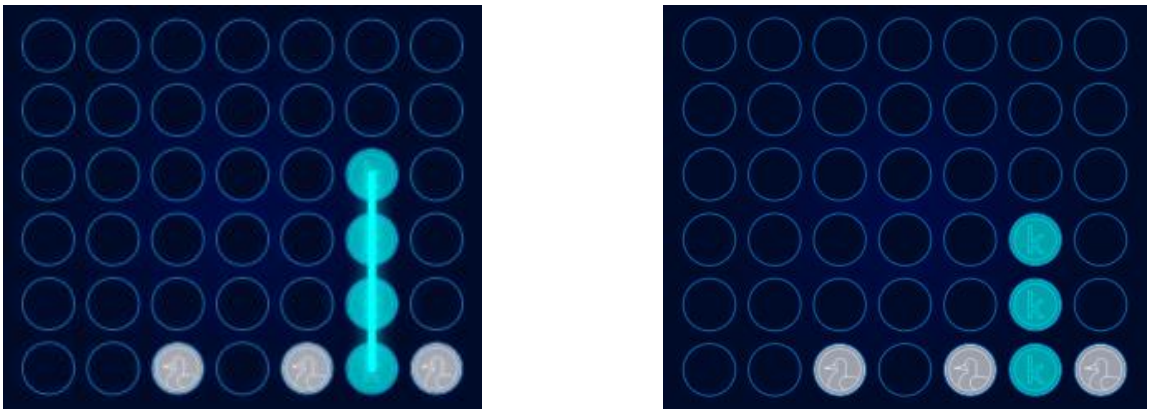
هر دو استراتژی پاداش دهی بالا را در نظر می‌گیریم و با هردو شبکه را آموزش می‌دهیم.

معماری شبکه: برای معماری شبکه دو راه پیش رو داریم. می‌توانیم از شبکه عادی چند لایه با Activation Function های RelU استفاده کنیم. همچنین می‌توانیم از چند لایه کانولوشنی در ابتدا استفاده نمائیم. در خصوص اضافه کردن Dropout، Batch-Normalization و یا Max-Pooling نیز با استفاده از آزمون و خطا تصمیم می‌گیریم.

تصویر ۲-۲ یک بازی کامل با حریف رندوم را نشان می‌دهد.

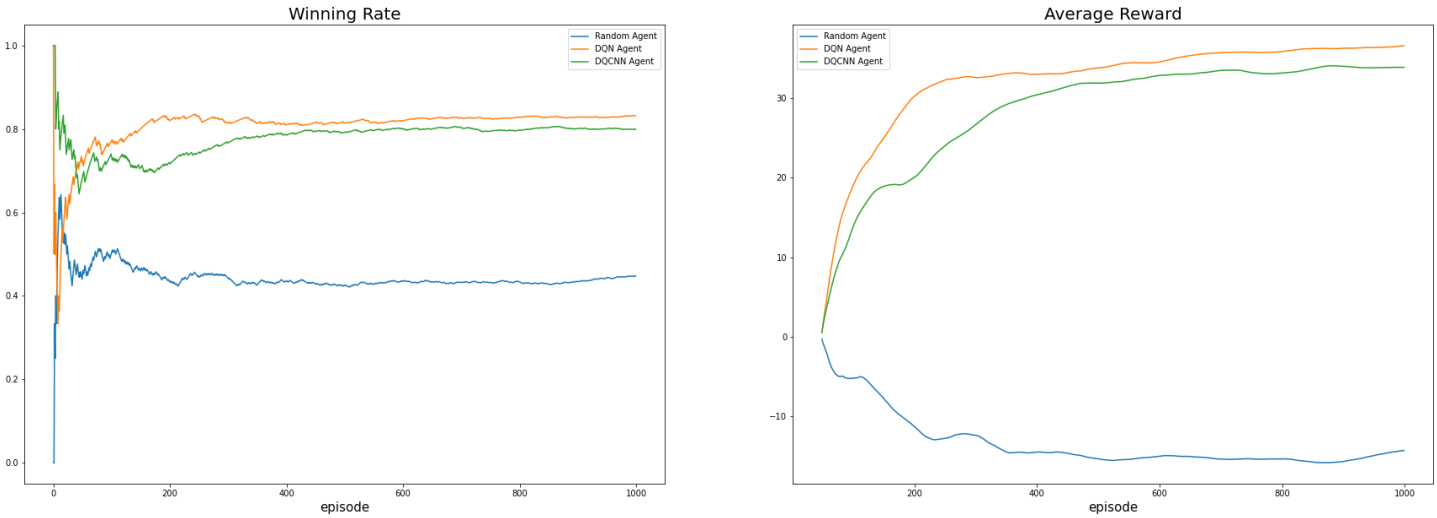






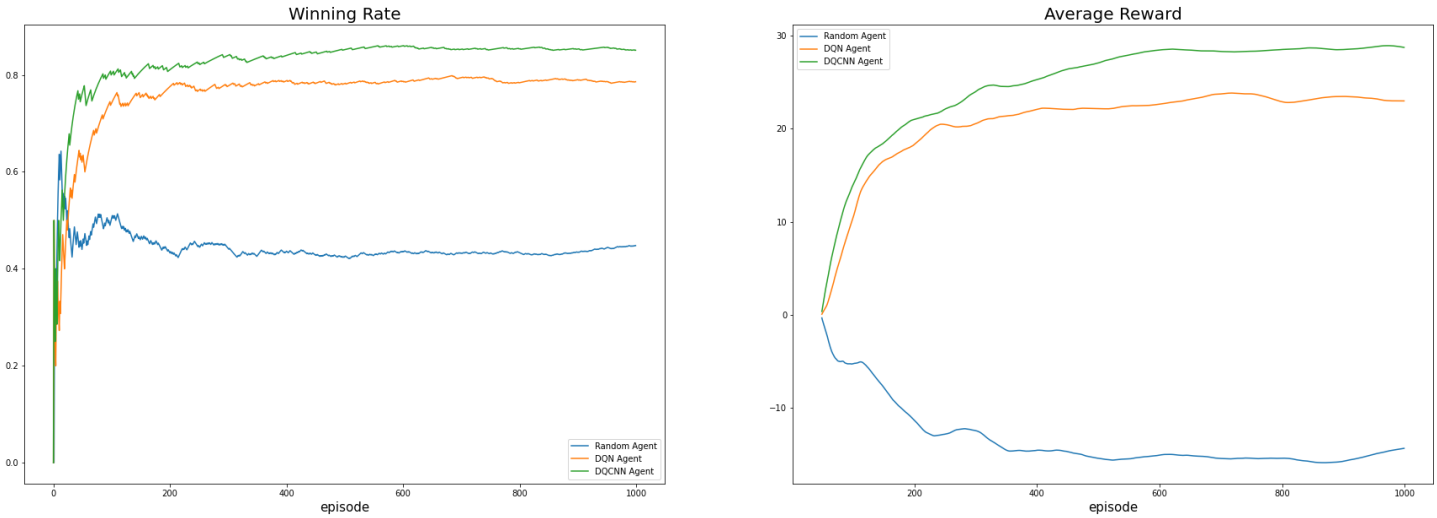
تصویر ۲-۲: یک بازی کامل با حریف رندوم

تصویر ۲-۳ نمودار های مربوط به یادگیری شبکه با استراتژی پاداش دهی اول را نشان می‌دهد.



تصویر ۲-۳: نمودار مربوط به یادگیری شبکه با استراتژی اول (پاداش هر حرکت $+1$)

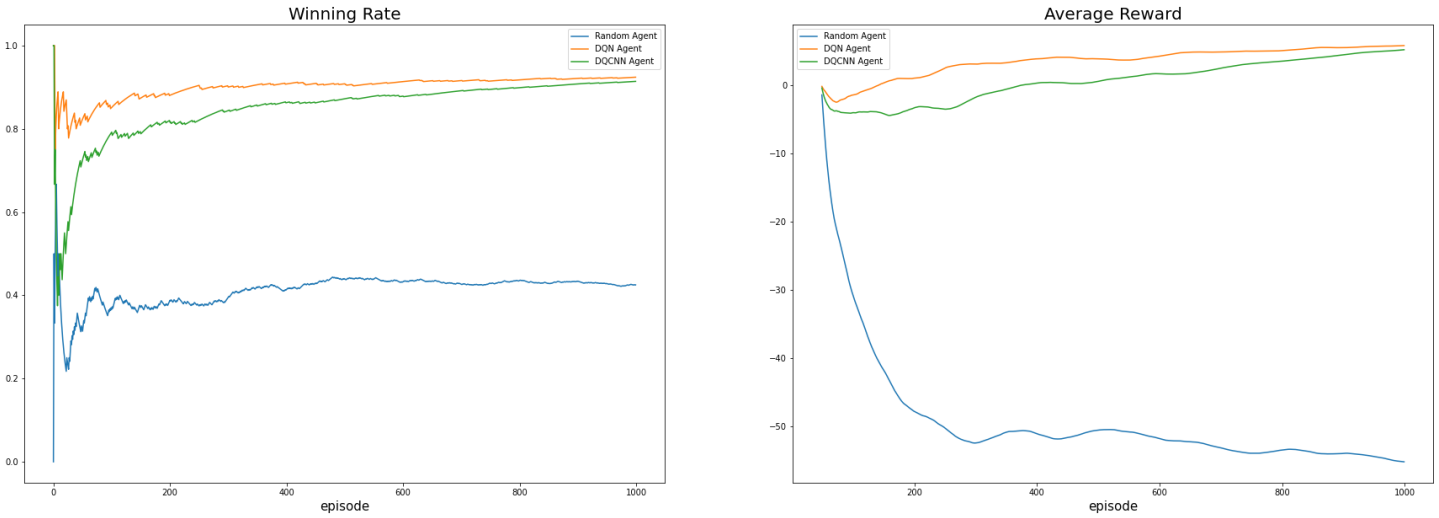
تصویر ۲-۴ نمودار های مربوط به یادگیری شبکه با استراتژی پاداش دهی اول را نشان می‌دهد.



تصویر ۲-۴: نمودار مربوط به یادگیری شبکه با استراتژی دوم (پاداش هر حرکت -1)

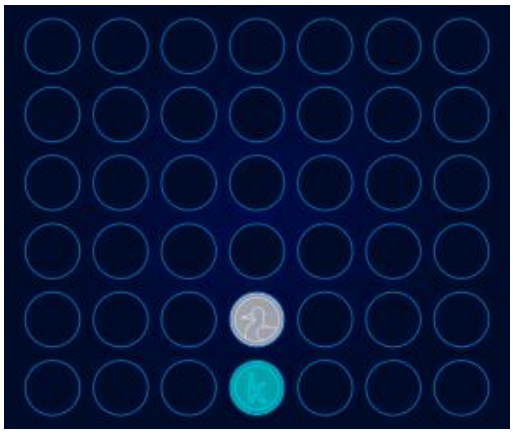
تحلیل نتایج: همانطور که در تصاویر بالا مشاهده می‌شود، هنگامی که از پاداش -1 به ازای هر حرکت استفاده می‌کنیم سرعت یادگیری بالاتر می‌رود. همچنین برای شبکه کانولوشنی، شاهد دقت بالاتری (پیروزی‌های بیشتری) هستیم. برای اینکه تعداد پیروزی‌ها افزایش یابد و نرخ پیروزی به 100% نزدیک شود، می‌توانیم از شبکه عمیق‌تر استفاده کنیم. اما یادگیری ای شبکه زمان‌بر تر خواهد بود.

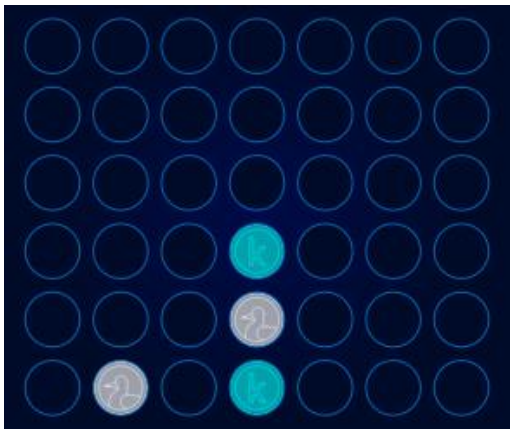
عیب یابی و راهکار بهبود نتایج: با مشاهده چند دور از بازی متوجه می‌شویم که Agent فقط به یک روش اتکا می‌کند و به بازی حریف توجهی ندارد. در نتیجه بهتر از اندازه جریمه شکست از پاداش پیروزی بیشتر شود. به این منظور پاداش پیروزی را $+20$ ، جریمه شکست را -100 و جریمه هر حرکت را -1 در نظر می‌گیریم. تصویر ۲-۵ نمودار های مربوط به این استراتژی را نشان می‌دهد.

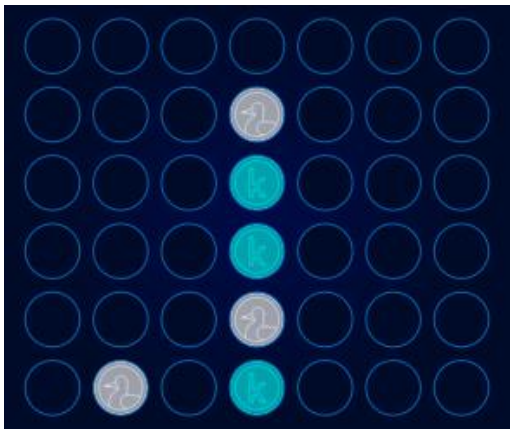


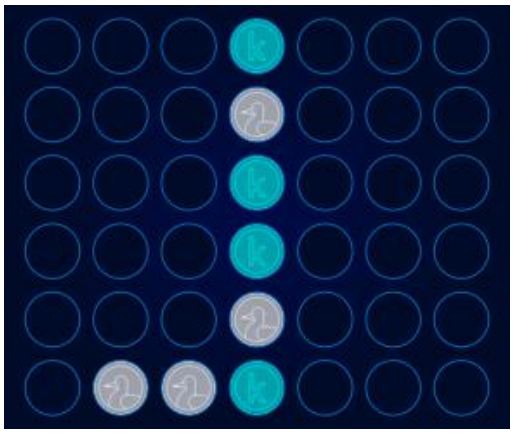
تصویر ۲-۵: نمودار مربوط به یادگیری شبکه با استراتژی بهبود یافته (پاداش پیروزی $+20$، جریمه شکست -100 و جریمه هر حرکت -1)

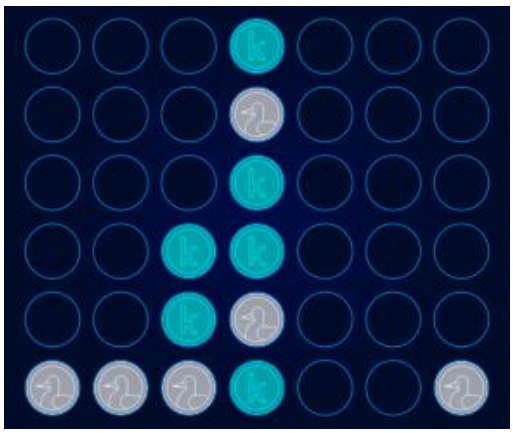
همچنین یک دور بازی با استراتژی جدید نیز در تصویر ۲-۶ نمایش داده شده است.

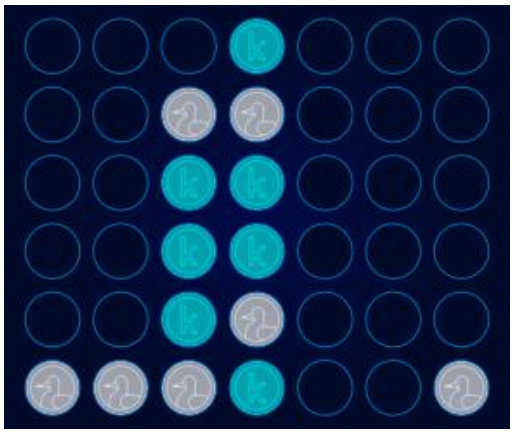


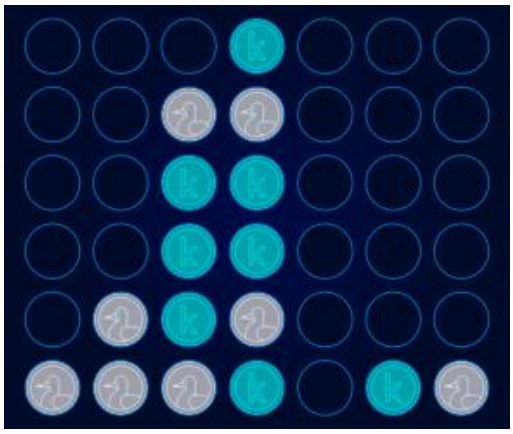


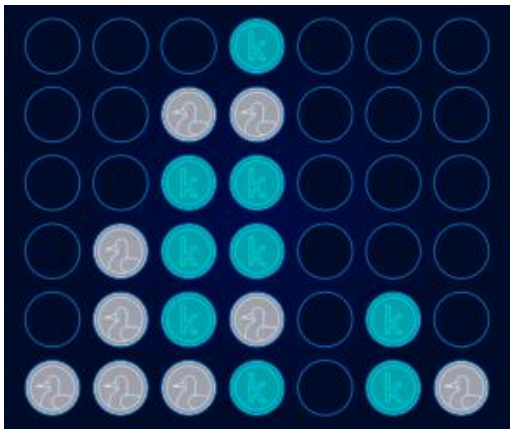


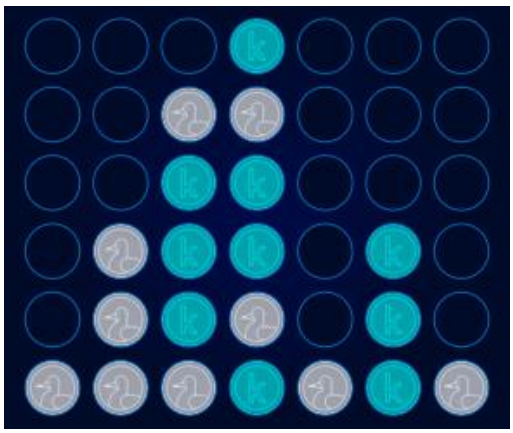


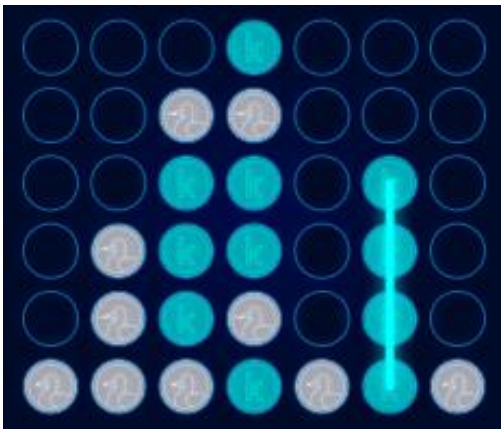












تصویر ۲-۶: نتیجه یک دور بازی با Agent آموزش دیده شده با استراتژی بهبود یافته (پاداش پیروزی $20+$، جریمه شکست $100-$ و جریمه هر حرکت $1-$)