Familiarity with RL-GANs and Deep Q-Learning, using them in simulating the connect 4 game

Golmehr Khosrokhavar

Student,ECE Department, Sharif University of Technology; University of Tehran, College of Engineering

*Corresponding Author: khosrokhavar.g@gmail.com

Abstract

GAN networks are capable of using distribution Possible input, input output, input output. But such networks are hard to train or During sustained training. The result in this paper is that before the generator block of the GAN network, an agent is arranged to give the correct input to the generator to generate the correct form of the input latent space. This factor eliminates the need for complex optimizations.

Introduction

In Figure 1, we can see a network that can complete an incomplete picture. for this At first, the Auto Encoder in the network should be trained to capture the latent space of the photo. Then, the network agent is trained to choose the best z vector from this space as a movement, and the GANA that has been trained in advance can produce a picture according to this vector. Unlike GAN networks that were trained using the back-propagation method to select the best z vector to produce the best and closest image to the input, this network works in real-time and also If the missing areas are larger, it is still resistant. At the end of the network, just like GAN networks, one Separator 1 is placed to select the best one between the output of the Auto Encode decoder and the decoded output of the GAN network and display it as the overall output. It is worth noting if the missing areas of the image is low, an Auto Encoder can display the correct image to an acceptable extent.

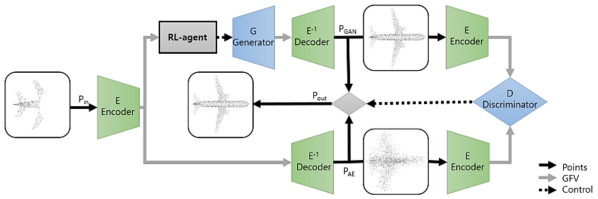

Figure 1: Structure of image complementing network

Introduction to RL-GANs

Adding the interactive learning factor solves the instability problem of current deep networks. The article also mentioned that training GAN networks on GFV (global feature vectors) can improve training stability. In Figure 2, we can see the l-GAN section, where the generating part produces a GFV from the noise according to the selected vector z. Then they are converted into a series of three-dimensional points for output by the Auto Encoder decoder. According to the combination of the following error functions, the reinforcement learning agent finds the best z for The GAN input selects:

- Chamfer error function that calculates the chamfer distance between the input pin

points and the generated points.

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

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

- GFV error function that calculates the second norm of the input GFV and z of the selected GFVs.

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

- Discriminator error function, which is its output

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

The environment is a combination of Auto Encoder and l-GAN for the interactive learning agent. Mode is the GFV of imperfect image noises. The best decision is the best seed selected to generate the generative random noise. Because different actions for this structure are continuous, a Deep deterministic policy gradient has been used, which means that an agent $\mu(s \mid \theta^{\mu})$ has learned a specific policy and assigns states to actions A predefined method maps this agent to the method The mathematical expectation of getting the following cost function is taught.

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

Finally, instead of optimizing the combination of error functions, the error functions are converted into rewards. The speed of convergence and finding the answer (completing the picture) is much higher, and the model can complete pictures up to about 70% of the missing data.

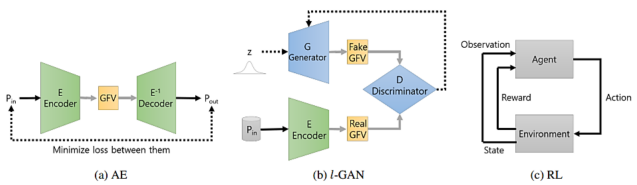

Figure 2: Three basic parts of RL-GAN-NET network

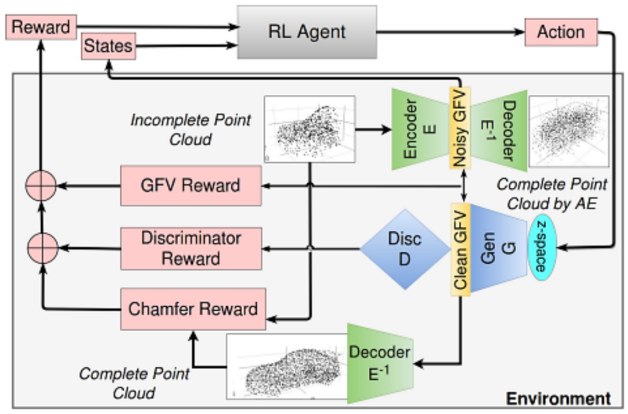

Figure 3: RL-GAN-Net training for picture completion

Algorithm description

Agent Input: State $(s_t) : s_t = GFV_n = \mathbf{E}(P_{in})$; Sample pointcloud P_{in} from dataset into the pre-trained encoder $\mathbf{E}$ 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 $\mathbf{G}$ to form clean latent vector $GFV_c = \mathbf{G}(z)$

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

1. Initialize procedure **Env** with pre-trained generator $\mathbf{G}$, discriminator $\mathbf{D}$, encoder $\mathbf{E}$ and decoder $\mathbf{E}^{-1}$
2. Initialize policy π with DDPG, actor $\mathbf{A}$, critic $\mathbf{C}$, and replay buffer $\mathbf{R}$
3. for $t_{\text{steps}} <$ maxsteps do Get P_{in}
5. if $t_{\text{steps}} > 0$ then Train $\mathbf{A}$ and $\mathbf{C}$ with $\mathbf{R}$
6. if $t_{\text{LastEvaluation}} > f_{\text{EvalFrequency}}$ then
. Evaluate π
7. $GFV_n \leftarrow \mathbf{E}(P_{\text{in}})$
8. if $t_{\text{steps}} > t_{\text{StartTime}}$ then
. Random Action a_t
9. if $t_{\text{steps}} < t_{\text{StartTime}}$ then
. Use $a_t \leftarrow \mathbf{A} \leftarrow GFV_n$
. $(s_t, a_t, r_t, s_{t+1}) \leftarrow \mathbf{Env} \leftarrow a_t$
. Store transition (s_t, a_t, r_t, s_{t+1}) in $\mathbf{R}$

endfor
10. procedure ENv (P_{in}, a_t)
11. Get State $(s_t) : GFV_n \leftarrow \mathbf{E}(P_{in})$

12. Implement Action : $GFV_c \leftarrow \mathbf{G}\,(a_t = z)$
13. Calculate reward r_t using Eq. (5)
14. Obtain point cloud : $P_{\text{out}} \leftarrow \mathbf{E}^{-1}\,(GFV_c)$

At first, we initialize the environment with a pre-trained generator, separator encoder, and decoder. Then, the interactive learning agent that includes the policy π is initialized by the DDPG method. Also, for actor-network (A), critic network (C), and responder buffer (R) also does this.
Takes a number of samples from the image points. In a positive interval, it trains actor and critic networks by buffer. It evaluates the policy in a time interval that is greater than the predefined EvalFrequency frequency.
The encoder encodes the sampled points of the image into the noise feature space (GFV). At If this time has passed since the start time, the agent performs a random movement (a_t); But if from Let's not pass the start time; the actor-network determines a movement for the agent according to the noise feature space (GFV_n) By giving specific actions a_t and with the help of the step function from Env environment, state, next state, reward, and movement information are determined. All step outputs are stored in buffer R.
The Env environment implements the following actions:

- It obtains the state s_t by encoding the input points into the noise feature space (GFV_n) by the pre-trained encoder.
- The generative network implements the action by producing a feature space without noise (GFV_c) according to the action $a_t = z$
- It calculates the reward according to the equation r = $w_{CH}{\cdot}r_{CH}+w_{GFV}{\cdot}r_{GFV}+w_D{\cdot}r_D$
- It is pre-trained by the decoder and obtains (GFV_c) output points by decoding the feature space without noise.

In simple words, the algorithm can be described as if the generating network acts like an agent and performs an action or movement in the environment, and its ultimate goal is to maximize the amount of reward. The separating network also returns the reward signal to the generating network each time. Encoder and decoder sections, respectively It takes the input points to the feature space and converts them to output points from the feature space.

Implementation of Connect-4 game using Deep Q-learning

Introduction:

Describe the design and operation of deep neural networks in predicting optimal actions. What is the basis for determining the loss function of these networks?
In Q-based deep learning networks, the input is the same states and the output is the predicted Q value for each action.
The design of networks is generally multi-layered and fully connected, depending on the complexity or simplicity of the problem, the number of layers and the number of neurons in each layer can be changed. The nonlinear activation functions tanh and ReLU are used to include the nonlinear part of the problem.
The function of such networks is to select the action with the highest Q value for the current state can be During training, the network is periodically updated based on the rewards it receives and the difference between the predicted Q value and the observed Q value.
The loss function of these networks is usually in the form of mean squared error between the predicted Q and the observed Q values. It should be noted that the observed Q values are calculated as the sum of current rewards and include the discount factor for future rewards. This type of loss function encourages the network, which predicts the value of Q that maximizes the expected reward for the action to private.

Implementation section

To implement the Connect4 game, one of the important parts is defining the reward values for each part of the game. If the agent wins the game, he gets a bonus of 50; If he loses, he will be fined -50. If the game ends without having

any brands, the agent is subject to a penalty of -50. It should be noted that by placing the dice in the game, the agent receives a bonus equal to 1/42 (42 game houses). The game's goal is that the agent wins most during many episodes and long playing sessions.
The agent uses the epsilon-greedy algorithm with an initial value of 1. During the episodes, it is increased by the coefficient 0.995 is reduced and repeats these actions until it reaches 0.01. The default value of the Adam learning parameter, which is 0.001, was used for the learning parameter.
Since the opponent plays randomly, paying attention only to the agent's reward during the game is impossible. Therefore, we also randomly compare the number of wins to the game.
For the agent to have the best performance, both parties must be agents themselves, but this takes a lot of time due to the large volume of calculations and the increase of game modes. For this reason, The opponent is used randomly to train this agent.

Result

For its neural network part, we first use 3 Dense layers; each layer has 32 neurons.

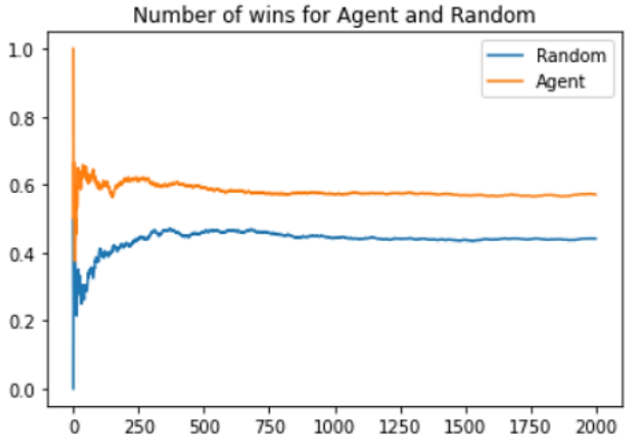

Figure 4: The number of wins (divided by the total number of games) for the model and randomly

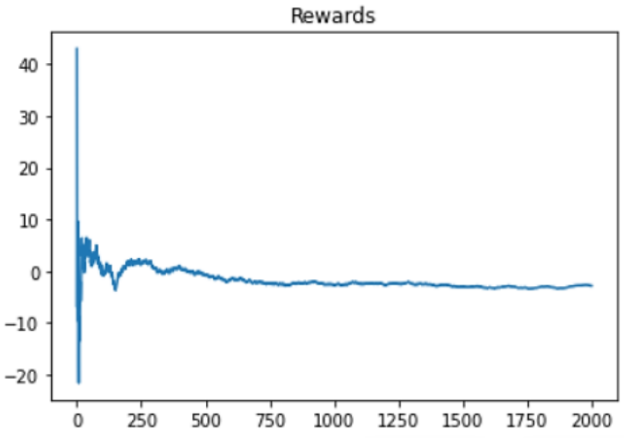

Figure 5: Rewards received by the model during the game

Then we increase the number of neurons in each layer to 64 and see the results:

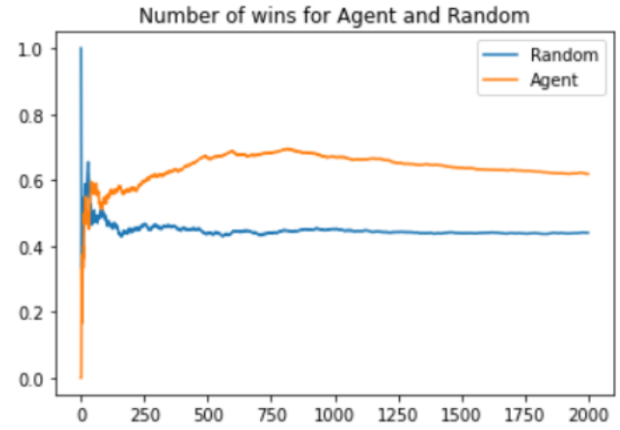

Figure 6: The number of wins (divided by the total number of games) randomly

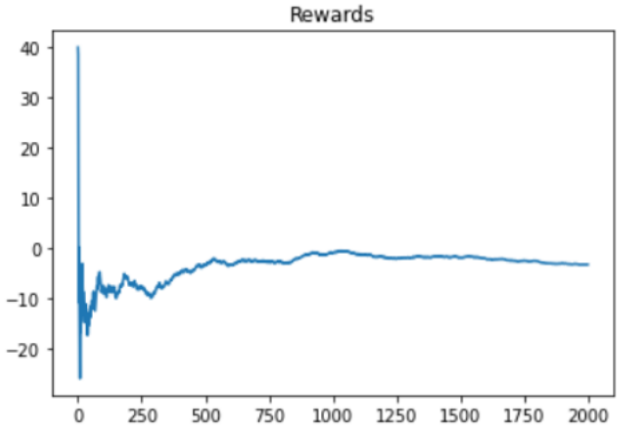

Figure 7: Rewards received by the model during the game

Result

Now we can see how the network results will change if we add a convolution layer to the layers. By placing a two-dimensional Conv layer with a filter size of 5x5 and several 32, we can see the results:

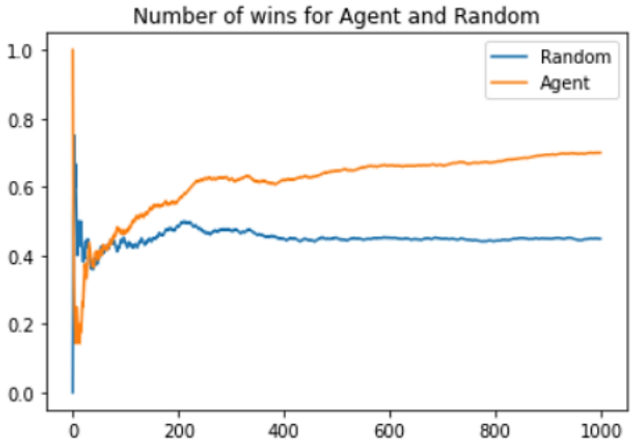

Figure 8: The number of wins (divided by the total number of games) for the model and randomly

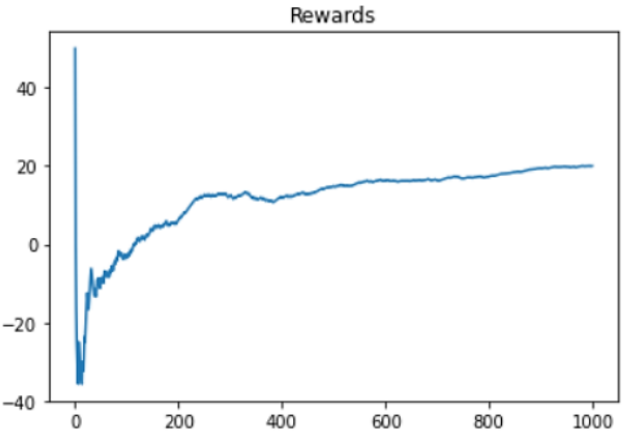

Figure 9: Rewards received by the model during the game

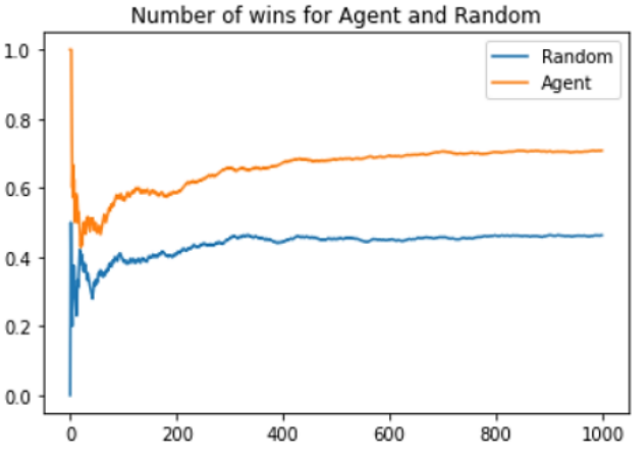

Figure 10: The number of wins (divided by the total number of games) for the model and randomly

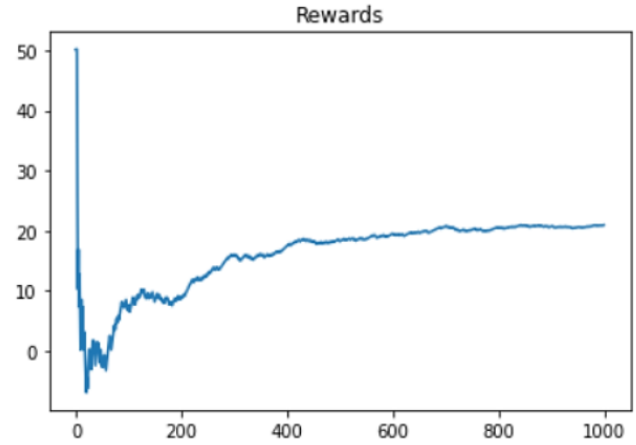

Figure 11: Rewards received by the model during the game

Result

Now, just in a two-dimensional Conv layer, we reduce the size of the filter to 3x3 and increase their number to 64:

So far we have seen the best result, that is, the best number of wins in 1000 episodes. To improve the results, it was decided to change the penalty amount for placing each piece in the game to -1. Now we see the same results in order.

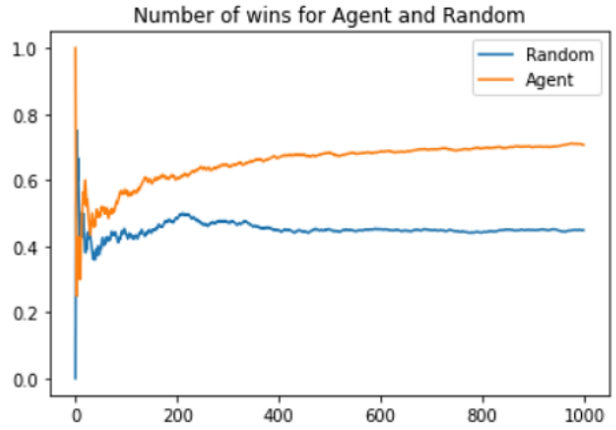

Figure 12: The number of wins (divided by the total number of games) for the model and randomly

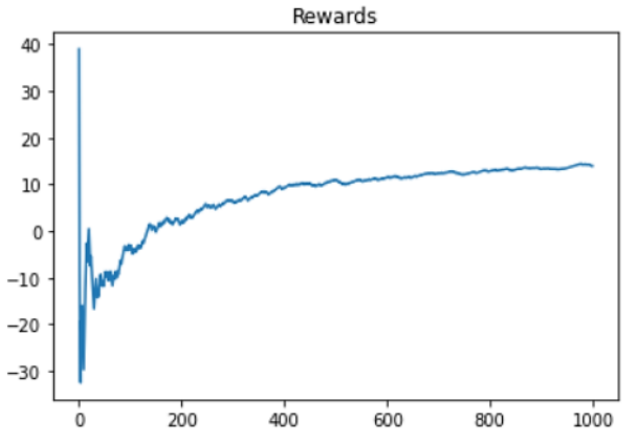

Figure 13: Rewards received by the model during the game

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

Figure 14: The architecture of the deep part of the network

It can be seen in Figures 12 and 13 that the best results of the implementation of the Connect4 game are for placing a 2D convolutional layer with a filter size of 3*3 and the number of 64, then 3 Dense layers with the number of 64 neurons in the network structure. If the agent wins the game, he gets a bonus of 50; If he loses, he will be fined -50; If the game ends without having any brands, the agent is subject to a penalty of -50 Also, the movement penalty is -1.

References

. 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
https://www.voigtstefan.me/post/connectx/
https://www.kaggle.com/code/phunghieu/connectx-with-q-learning/notebook
https://keon.github.io/deep-q-learning/