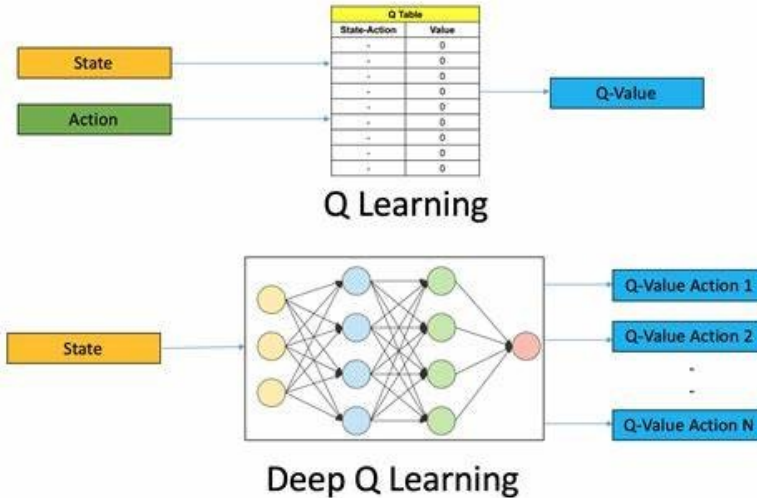


Deep Reinforcement Learning

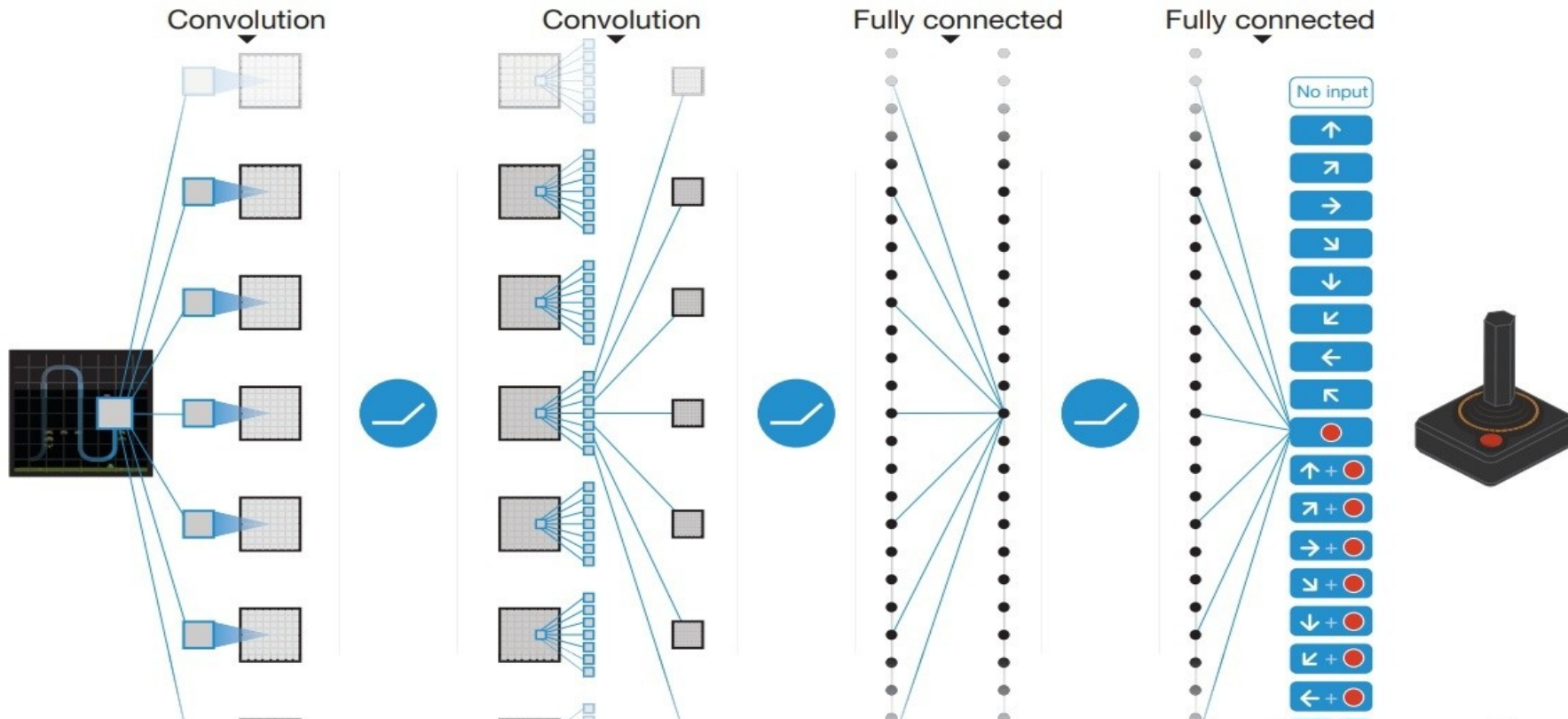


Deep Q-Network

DQN 为基于深度学习的 Q-learning 算法，而在 Q-learning 中，使用表格来存储每一个 state 下 action 的 reward，即状态 - 动作值函数 $Q(s, a)$ 。但是在实际任务中，状态量通常数量巨大并且在连续的任务中，会遇到维度灾难的问题，所以使用真正的 Value Function 通常是不切实际的，所以使用了价值函数近似（Value Function Approximation）的表示方法。

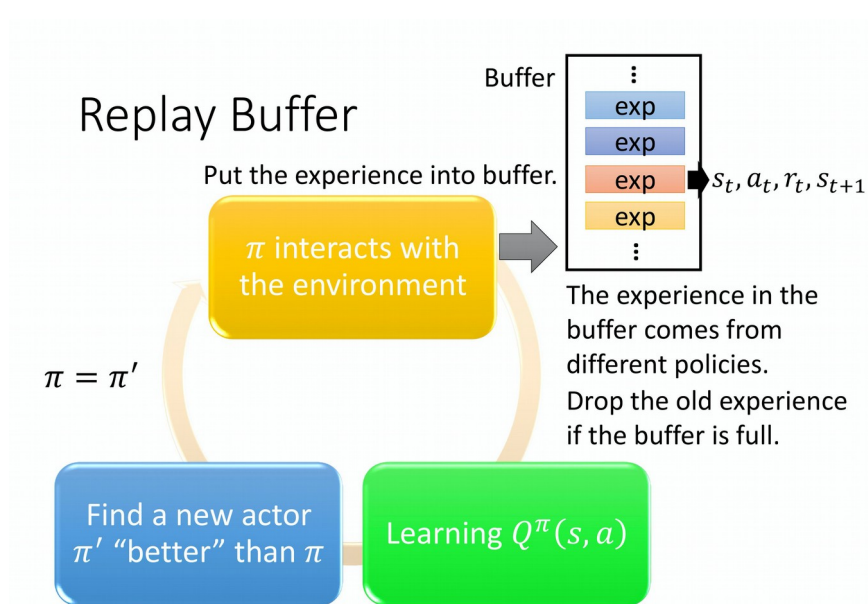
DQN 是在 Q-learning 算法基础上采取函数逼近的方法来估计状态值函数，并且有两个主要的创新点：经验回放 (replay buffer)，以及目标网络 (target network)

Deep Q-Network



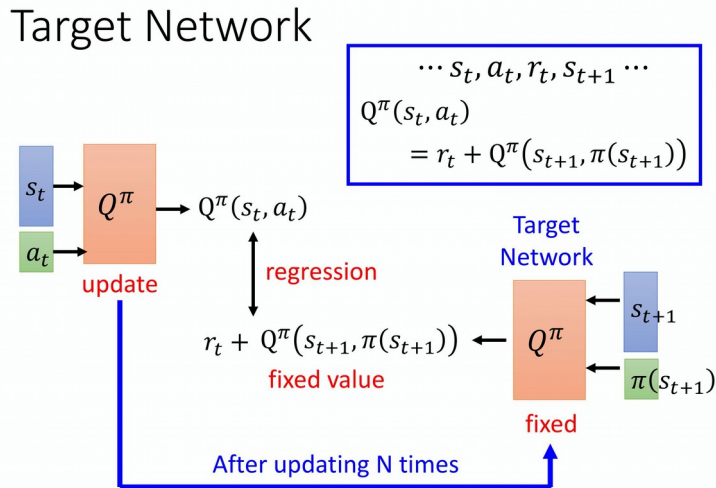
经验回放

Replay Buffer 又被称为 Replay Memory。Replay Buffer 是说现在会有某一个策略 π 去跟环境做互动，然后它会去收集数据。我们会把所有的数据放到一个 buffer 里面，buffer 里面就存了很多数据。



目标网络

Target Network：为了解决在基于 TD 的 Network 的问题时，优化目标 $Q^\pi(s_t, a_t) = r_t + Q^\pi(s_{t+1}, \pi(s_{t+1}))$ 左右两侧会同时变化使得训练过程不稳定，从而增大 regression 的难度。target network 选择将上式的右部分即 $r_t + Q^\pi(s_{t+1}, \pi(s_{t+1}))$ 固定，通过改变上式左部分的 network 的参数，进行 regression。



目标网络

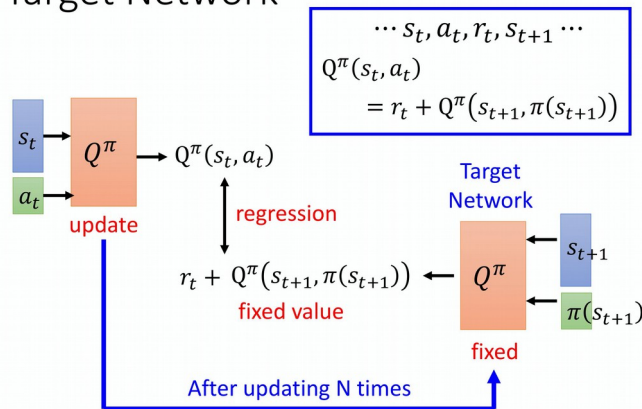
在训练的时候，右边 Q 网络固定住，只更新左边的 Q 网络的参数。因为右边的 Q 网络负责产生目标，所以叫目标网络。因为目标网络是固定的，所以现在得到的目标 $r_t + Q^\pi(s_{t+1}, \pi(s_{t+1}))$ 的值也是固定的。因为目标网络是固定的，只调左边网络的参数，就变成是一个回归问题。在实现的时候，会把左边的 Q 网络更新好几次以后，再去用更新过的 Q 网络替换这个目标网络。

$$Q^\pi(s_t, a_t) = r_t + Q^\pi(s_{t+1}, \pi(s_{t+1}))$$

模型的输出 estimation

目标 target

Target Network



Algorithm 1: deep Q-learning with experience replay.

Initialize replay memory D to capacity N

数据池D

Initialize action-value function Q with random weights θ

初始化两个网络Q Q'

Initialize target action-value function \hat{Q} with weights $\theta^- = \theta$

For episode = 1, M **do**

Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$

For $t = 1, T$ **do**

With probability ε select a random action a_t

贪婪算法选择action

otherwise select $a_t = \operatorname{argmax}_a Q(\phi(s_t), a; \theta)$

Execute action a_t in emulator and observe reward r_t and image x_{t+1}

Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$

Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D

存储数据到D

Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D

从D中随机选取batch_size条数据

Set $y_j = \begin{cases} r_j & \text{if episode terminates at step } j+1 \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^-) & \text{otherwise} \end{cases}$

执行梯度下降, 更新参数

Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ with respect to the network parameters θ

Every C steps reset $\hat{Q} = Q$

每隔C更新Q'

End For

End For

Deep Q-Network Agents

The deep Q-network (DQN) algorithm is a model-free, online, off-policy reinforcement learning method. A DQN agent is a value-based reinforcement learning agent that trains a critic to estimate the return or future rewards. DQN is a variant of Q-learning.

DQN agents can be trained in environments with the following observation and action spaces.

Observation Space	Action Space
Continuous or discrete	Discrete

Deep Q-Network Agents

DQN agents use the following critic.

Critic	Actor
Q-value function critic $Q(S,A)$, which you create using <code>rlQValueFunction</code> or <code>rlVectorQValueFunction</code>	DQN agents do not use an actor.

Deep Q-Network Agents

During training, the agent:

- Updates the critic properties at each time step during learning.
- Explores the action space using epsilon-greedy exploration. During each control interval, the agent either selects a random action with probability ϵ or selects an action greedily with respect to the value function with probability $1-\epsilon$. This greedy action is the action for which the value function is greatest.
- Stores past experiences using a circular experience buffer. The agent updates the critic based on a mini-batch of experiences randomly sampled from the buffer.

Critic Function Approximator

To estimate the value function, a DQN agent maintains two function approximators:

- Critic $Q(S,A;\phi)$ — The critic, with parameters ϕ , takes observation S and action A as inputs and returns the corresponding expectation of the long-term reward.
- Target critic $Q^t(S,A;\phi_t)$ — To improve the stability of the optimization, the agent periodically updates the target critic parameters ϕ_t using the latest critic parameter values.

Both $Q(S,A;\phi)$ and $Q^t(S,A;\phi_t)$ have the same structure and parameterization.

During training, the agent tunes the parameter values in ϕ . After training, the parameters remain at their tuned value and the trained value function approximator is stored in critic $Q(S,A)$.

Agent Creation

You can create and train DQN agents at the MATLAB® command line or using the Reinforcement Learning Designer app. At the command line, you can create a DQN agent with a critic based on the observation and action specifications from the environment. To do so, perform the following steps.

1. Create observation specifications for your environment. If you already have an environment interface object, you can obtain these specifications using `getObservationInfo`.
2. Create action specifications for your environment. If you already have an environment interface object, you can obtain these specifications using `getActionInfo`.
3. If needed, specify the number of neurons in each learnable layer or whether to use an LSTM layer. To do so, create an agent initialization option object using `rlAgentInitializationOptions`.
4. If needed, specify agent options using an `rlDQNAgentOptions` object.
5. Create the agent using an `rlDQNAgent` object.

Agent Creation

Alternatively, you can create actor and critic and use these objects to create your agent. In this case, ensure that the input and output dimensions of the actor and critic match the corresponding action and observation specifications of the environment.

1. Create a critic using an `rlQValueFunction` object.
2. Specify agent options using an `rlDQNAgentOptions` object.
3. Create the agent using an `rlDQNAgent` object.

DQN agents support critics that use recurrent deep neural networks as functions approximators.

Training Algorithm

DQN agents use the following training algorithm, in which they update their critic model at each time step. To configure the training algorithm, specify options using an `rlDQNAgentOptions` object.

Initialize the critic $Q(s,a;\phi)$ with random parameter values ϕ , and initialize the target critic parameters ϕ_t with the same values. $\phi_t = \phi$.

For each training time step:

1. For the current observation S , select a random action A with probability ϵ . Otherwise, select the action for which the critic value function is greatest.
$$A = \arg \max_A Q(S, A; \phi)$$

Training Algorithm

2. Execute action A . Observe the reward R and next observation S' .
3. Store the experience (S,A,R,S') in the experience buffer.
4. Sample a random mini-batch of M experiences (S_i,A_i,R_i,S'_i) from the experience buffer. To specify M , use the `MiniBatchSize` option.
5. If S'_i is a terminal state, set the value function target y_i to R_i . Otherwise, set it to

$$A_{\max} = \arg \max_{A'} Q(S'_i, A'; \phi) \quad (\text{double DQN})$$

$$y_i = R_i + \gamma Q_t(S'_i, A_{\max}; \phi_t)$$

$$y_i = R_i + \gamma \max_{A'} Q_t(S'_i, A'; \phi_t) \quad (\text{DQN})$$

To set the discount factor γ , use the `DiscountFactor` option. To use double DQN, set the `UseDoubleDQN` option to `true`.

Training Algorithm

6. Update the critic parameters by one-step minimization of the loss L across all sampled experiences.

$$L = \frac{1}{M} \sum_{i=1}^M (y_i - Q(S_i, A_i; \phi))^2$$

7. Update the target critic parameters depending on the target update method. For more information, see [Target Update Methods](#).
8. Update the probability threshold ϵ for selecting a random action based on the decay rate you specify in the `EpsilonGreedyExploration` option.

Target Update Methods

DQN agents update their target critic parameters using one of the following target update methods.

- Smoothing — Update the target parameters at every time step using smoothing factor τ . To specify the smoothing factor, use the TargetSmoothFactor option.

$$\phi_t = \tau\phi + (1 - \tau)\phi_t$$

- Periodic — Update the target parameters periodically without smoothing (TargetSmoothFactor = 1). To specify the update period, use the TargetUpdateFrequency parameter.
- Periodic Smoothing — Update the target parameters periodically with smoothing.

Target Update Methods

To configure the target update method, create a `rlDQNAgentOptions` object, and set the `TargetUpdateFrequency` and `TargetSmoothFactor` parameters as shown in the following table.

Update Method	TargetUpdateFrequency	TargetSmoothFactor
Smoothing (default)	1	Less than 1
Periodic	Greater than 1	1
Periodic smoothing	Greater than 1	Less than 1

DRL 示例

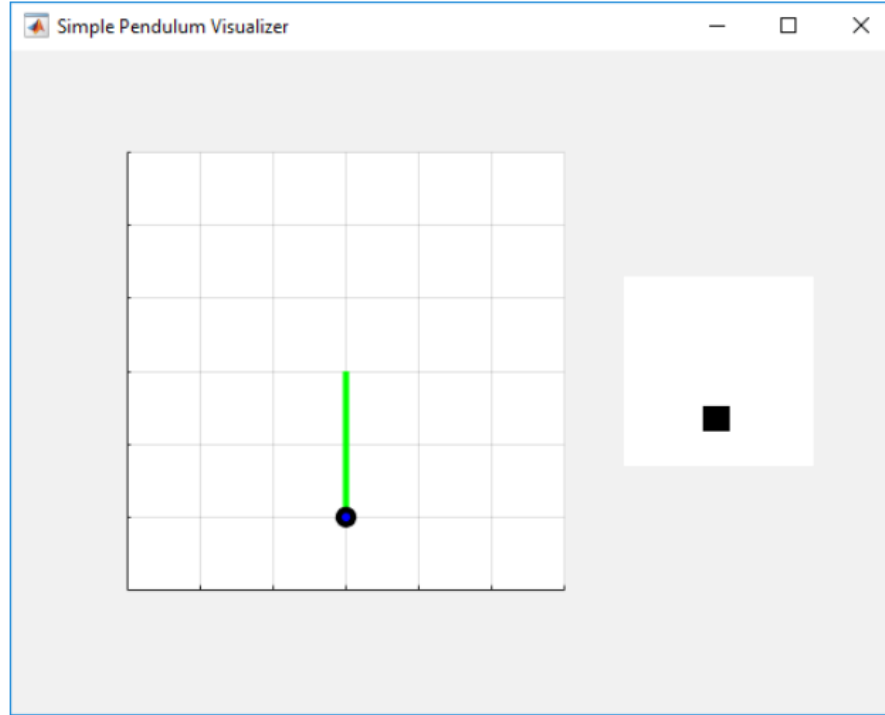
- Create Agent Using Deep Network Designer and Train Using Image Observations

Create Agent Using Deep Network Designer and Train Using Image Observations

This example shows how to create a deep Q-learning network (DQN) agent that can swing up and balance a pendulum modeled in MATLAB®. In this example, you create the DQN agent using Deep Network Designer.

Pendulum Swing-Up with Image MATLAB Environment

The reinforcement learning environment for this example is a simple frictionless pendulum that initially hangs in a downward position. The training goal is to make the pendulum stand upright without falling over using minimal control effort.



Pendulum Swing-Up with Image MATLAB Environment

For this environment:

- The upward balanced pendulum position is 0 radians, and the downward hanging position is pi radians.
- The torque action signal from the agent to the environment is from -2 to $2 \text{ N} \cdot \text{m}$.
- The observations from the environment are the simplified grayscale image of the pendulum and the pendulum angle derivative.
- The reward , provided at every time step, is

$$r_t = -(\theta_t^2 + 0.1\dot{\theta}_t^2 + 0.001u_{t-1}^2)$$

Here:

- θ_t is the angle of displacement from the upright position.
- $\dot{\theta}_t$ is the derivative of the displacement angle.
- u_{t-1} is the control effort from the previous time step.

Create Environment Interface

Create a predefined environment interface for the pendulum.

```
env = rlPredefinedEnv('SimplePendulumWithImage-Discrete');
```

The interface has two observations. The first observation, named "pendImage", is a 50-by-50 grayscale image.

```
obsInfo = getObservationInfo(env);
```

```
obsInfo(1)
```

```
ans =
```

```
rlNumericSpec with properties:
```

```
LowerLimit: 0
```

```
UpperLimit: 1
```

```
Name: "pendImage"
```

```
Description: [0x0 string]
```

```
Dimension: [50 50]
```

```
DataType: "double"
```

Create Environment Interface

The second observation, named "angularRate", is the angular velocity of the pendulum.

obsInfo(2)

```
ans =  
  rlNumericSpec with properties:  
  
    LowerLimit: -Inf  
    UpperLimit: Inf  
    Name: "angularRate"  
    Description: [0x0 string]  
    Dimension: [1 1]  
    DataType: "double"
```


Create Environment Interface

The interface has a discrete action space where the agent can apply one of five possible torque values to the pendulum: -2 , -1 , 0 , 1 , or $2 \text{ N} \cdot \text{m}$.

```
actInfo = getActionInfo(env)
```

```
actInfo =
```

```
    rlFiniteSetSpec with properties:
```

```
    Elements: [-2 -1 0 1 2]
```

```
    Name: "torque"
```

```
    Description: [0x0 string]
```

```
    Dimension: [1 1]
```

```
    DataType: "double"
```

Fix the random generator seed for reproducibility.

```
rng(0)
```

Construct Critic Network Using Deep Network Designer

A DQN agent approximates the long-term reward, given observations and actions, using a critic value function representation. For this environment, the critic is a deep neural network with three inputs (two observations and one action), and one output.

You can construct the critic network interactively by using the Deep Network Designer app. To do so, you first create separate input paths for each observation and action. These paths learn lower-level features from their respective inputs. You then create a common output path that combines the outputs from the input paths.

Create Image Observation Path

To create the image observation path, first drag an `imageInputLayer` from the Layer Library pane to the canvas. Set the layer `InputSize` to 50,50,1 for the image observation, and set `Normalization` to none.

DESIGNER



New



Duplicate



Cut



Copy



Paste

Fit
to View

Zoom In



Zoom Out

Auto
Arrange

Analyze



Export

NETWORK

BUILD

NAVIGATE

LAYOUT

ANALYSIS

EXPORT

LAYER LIBRARY

Filter layers...

INPUT



imageInputLayer



image3dInputLayer



sequenceInputLayer



featureInputLayer



roiInputLayer

CONVOLUTION AND FULLY CONNECTED



convolution2dLayer



convolution3dLayer



groupedConvolution2dLayer



transposedConv2dLayer



transposedConv3dLayer



fullyConnectedLayer

Designer

Data

Training

PROPERTIES



imageInputLayer ?

Name	pendImage
InputSize	50,50,1
Normalization	none
NormalizationDimension	auto
Mean	[]
StandardDeviation	[]
Min	[]
Max	[]

OVERVIEW

Create Image Observation Path

Second, drag a `convolution2DLayer` to the canvas and connect the input of this layer to the output of the `imageInputLayer`. Create a convolution layer with 2 filters (`NumFilters` property) that have a height and width of 10 (`FilterSize` property), and use a stride of 5 in the horizontal and vertical directions (`Stride` property).

DESIGNER



New



Duplicate



Cut



Copy



Paste

Fit
to View

Zoom In



Zoom Out

Auto
Arrange

Analyze



Export

NETWORK

BUILD

NAVIGATE

LAYOUT

ANALYSIS

EXPORT

LAYER LIBRARY

Filter layers...

INPUT



imageInputLayer



image3dInputLayer



sequenceInputLayer



featureInputLayer



roiInputLayer

CONVOLUTION AND FULLY CONNECTED



convolution2dLayer



convolution3dLayer



groupedConvolution2dLayer



transposedConv2dLayer



transposedConv3dLayer



fullyConnectedLayer

Designer

Data

Training

 pendImage
imageInputLayer img_conv1
convolution2dL...

PROPERTIES



convolution2dLayer ?

Name	img_conv1
FilterSize	10,10
NumFilters	2
Stride	5,5
DilationFactor	1,1
Padding	same
Weights	[]
Bias	[]
WeightLearnRateFactor	1
WeightL2Factor	1
BiasLearnRateFactor	1

OVERVIEW

Create Image Observation Path

Finally, complete the image path network with two sets of `reLULayer` and `fullyConnectedLayer` layers. The output sizes of the first and second `fullyConnectedLayer` layers are 400 and 300, respectively.

Deep Network Designer

DESIGNER

New Duplicate Cut Copy Paste Fit to View Zoom In Zoom Out Auto Arrange Analyze Export

NETWORK BUILD NAVIGATE LAYOUT ANALYSIS EXPORT

LAYER LIBRARY

Filter layers...

transposedConv3dLayer

fullyConnectedLayer

SEQUENCE

IstmLayer

biIstmLayer

gruLayer

sequenceFoldingLayer

sequenceUnfoldingLayer

flattenLayer

wordEmbeddingLayer

ACTIVATION

reluLayer

leakyReluLayer

pendImage
imageInputLayer

img_conv1
convolution2dL...

img_relu
reluLayer

critic_theta_fc1
fullyConnected...

theta_relu1
reluLayer

critic_theta_fc2
fullyConnected...

PROPERTY

fullyConnectedLayer ?

Name critic_theta_fc2

InputSize auto

OutputSize 300

Weights []

Bias []

WeightLearnRateFactor 1

WeightL2Factor 1

BiasLearnRateFactor 1

BiasL2Factor 0

WeightsInitializer gloriot

BiasInitializer zeros

OVERVIEW

The screenshot displays the Deep Network Designer software interface. The top toolbar includes icons for New, Duplicate, Cut, Copy, Paste, Fit to View, Zoom In, Zoom Out, Auto Arrange, Analyze, and Export. Below the toolbar are tabs for NETWORK, BUILD, NAVIGATE, LAYOUT, ANALYSIS, and EXPORT. The main workspace shows a vertical sequence of layers: pendImage (imageInputLayer), img_conv1 (convolution2dL...), img_relu (reluLayer), critic_theta_fc1 (fullyConnected...), theta_relu1 (reluLayer), and critic_theta_fc2 (fullyConnected...). The Layer Library on the left lists various layers, with 'fullyConnectedLayer' and 'reluLayer' highlighted by red circles. The Properties panel on the right shows the configuration for the selected 'fullyConnectedLayer', with 'Name' set to 'critic_theta_fc2', 'OutputSize' set to '300', and other parameters like 'InputSize' set to 'auto'. The Overview section at the bottom right shows a visual representation of the network structure.

Create All Input Paths and Output Path

Construct the other input paths and output path in a similar manner. For this example, use the following options.

Angular velocity path (scalar input):

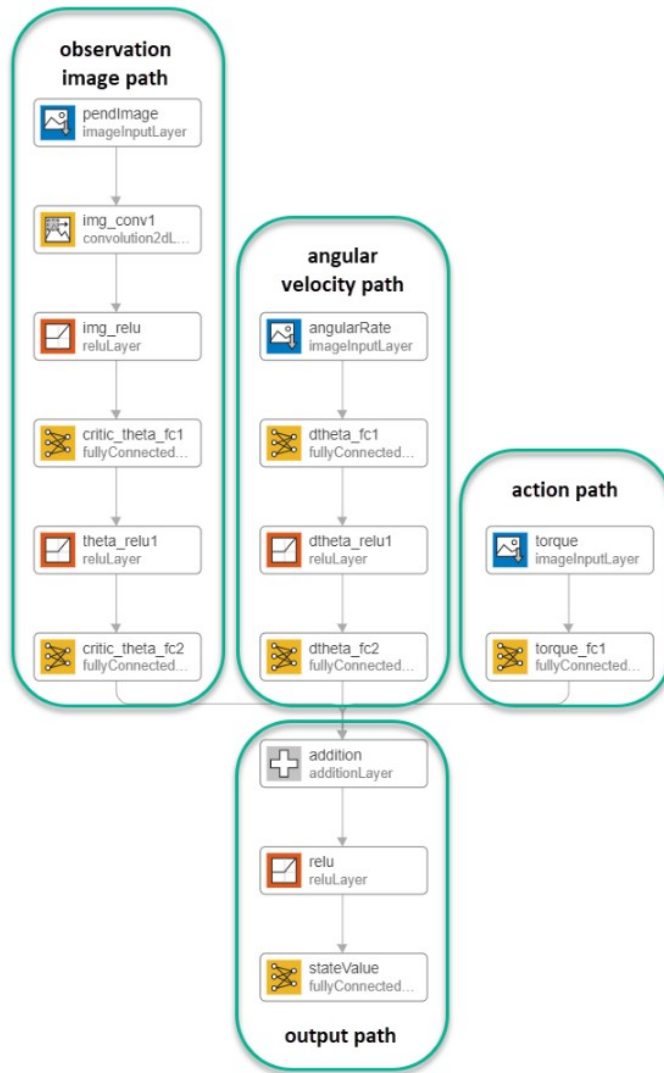
- `imageInputLayer` — Set `InputSize` to 1,1 and `Normalization` to none.
- `fullyConnectedLayer` — Set `OutputSize` to 400.
- `reLULayer`
- `fullyConnectedLayer` — Set `OutputSize` to 300.

Action path (scalar input):

- `imageInputLayer` — Set `InputSize` to 1,1 and `Normalization` to none.
- `fullyConnectedLayer` — Set `OutputSize` to 300.

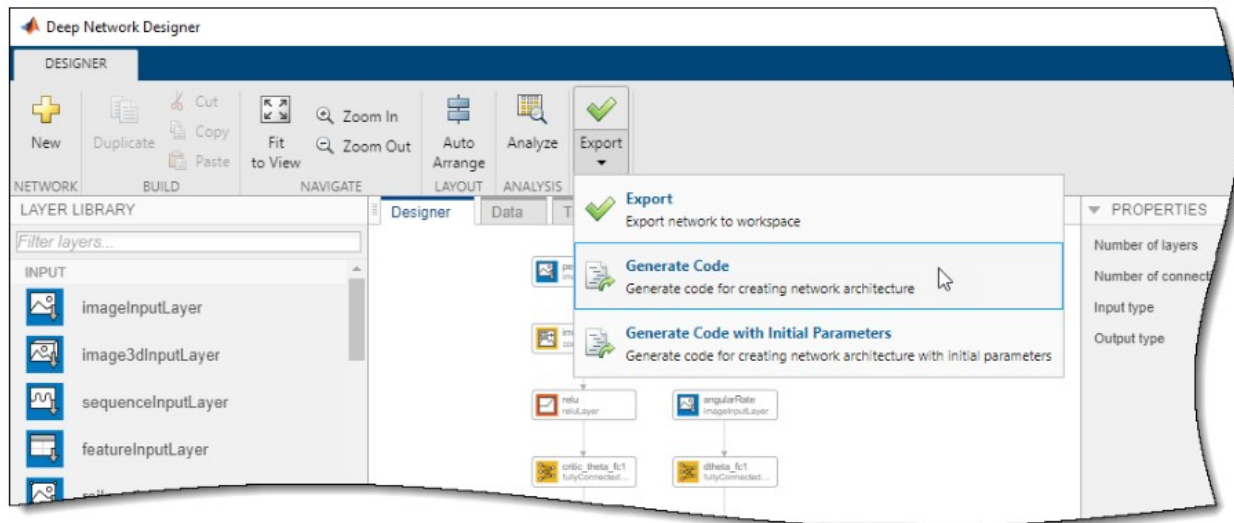
Output path:

- `additionLayer` — Connect the output of all input paths to the input of this layer.
- `reLULayer`
- `fullyConnectedLayer` — Set `OutputSize` to 1 for the scalar value function.



Export Network from Deep Network Designer

To export the network to the MATLAB workspace, in Deep Network Designer, click Export. Deep Network Designer exports the network as a new variable containing the network layers. You can create the critic representation using this layer network variable. Alternatively, to generate equivalent MATLAB code for the network, click Export > Generate Code.



```

lgraph = layerGraph();

templayers = [
    imageInputLayer([1 1 1], "Name", "angularRate", "Normalization", "none")
    fullyConnectedLayer(400, "Name", "dtheta_fc1")
    reluLayer("Name", "dtheta_relu1")
    fullyConnectedLayer(300, "Name", "dtheta_fc2")];
lgraph = addLayers(lgraph, templayers);

templayers = [
    imageInputLayer([1 1 1], "Name", "torque", "Normalization", "none")
    fullyConnectedLayer(300, "Name", "torque_fc1")];
lgraph = addLayers(lgraph, templayers);

templayers = [
    imageInputLayer([50 50 1], "Name", "pendImage", "Normalization", "none")
    convolution2dLayer([10 10], 2, "Name", "img_conv1", "Padding", "same", "Stride", [5 5])
    reluLayer("Name", "relu_1")
    fullyConnectedLayer(400, "Name", "critic_theta_fc1")
    reluLayer("Name", "theta_relu1")
    fullyConnectedLayer(300, "Name", "critic_theta_fc2")];
lgraph = addLayers(lgraph, templayers);

templayers = [
    additionLayer(3, "Name", "addition")
    reluLayer("Name", "relu_2")
    fullyConnectedLayer(1, "Name", "stateValue")];
lgraph = addLayers(lgraph, templayers);

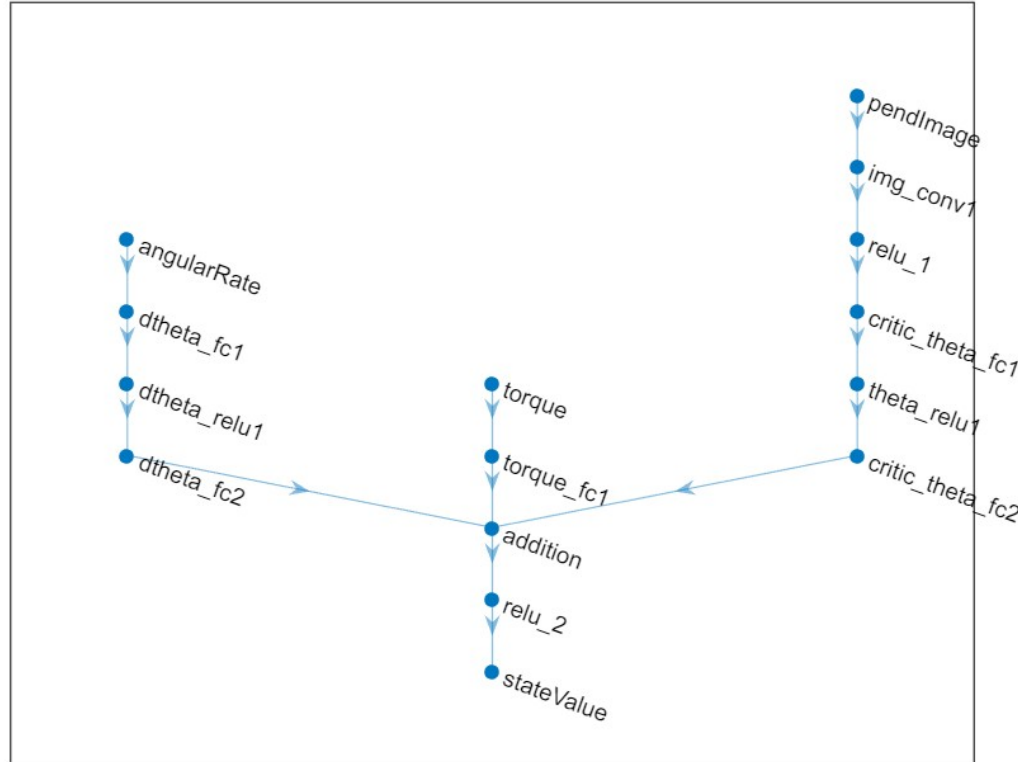
lgraph = connectLayers(lgraph, "torque_fc1", "addition/in3");
lgraph = connectLayers(lgraph, "critic_theta_fc2", "addition/in1");
lgraph = connectLayers(lgraph, "dtheta_fc2", "addition/in2");

```

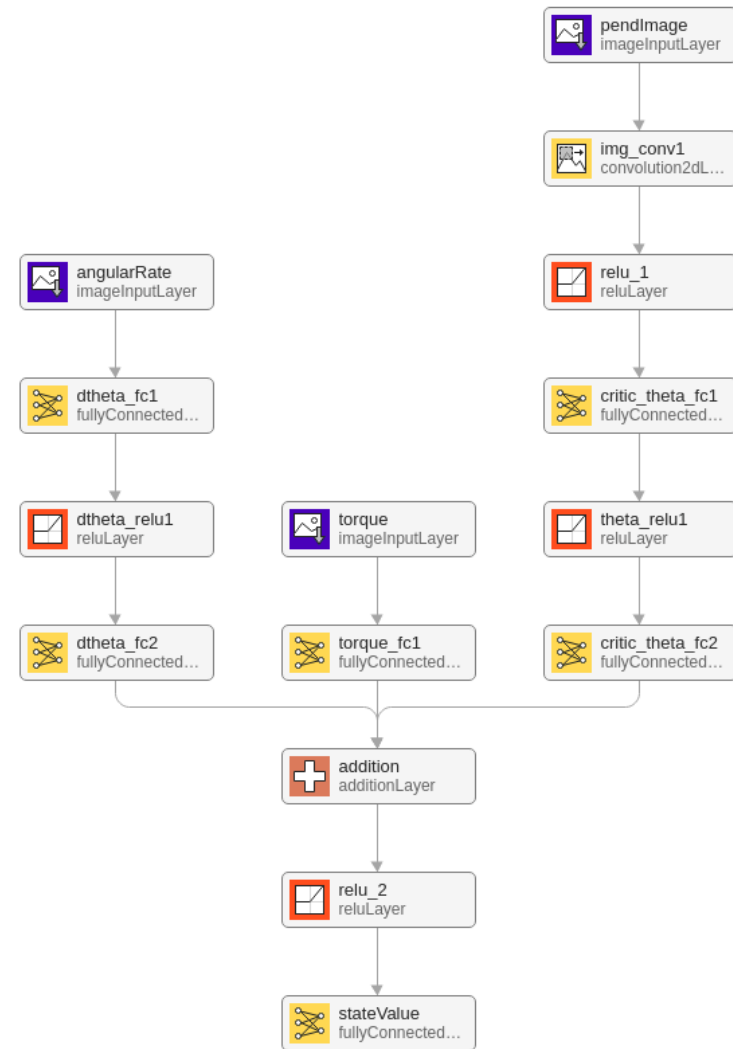
Export Network from Deep Network Designer

View the critic network configuration.

figure
plot(lgraph)



View Network from Deep Network Designer



深度网络设计器中的训练分析

名称: 来自深度网络设计器的网络

分析日期: 2022-12-03 09:21:19

322.9k

可学习参数总数

15

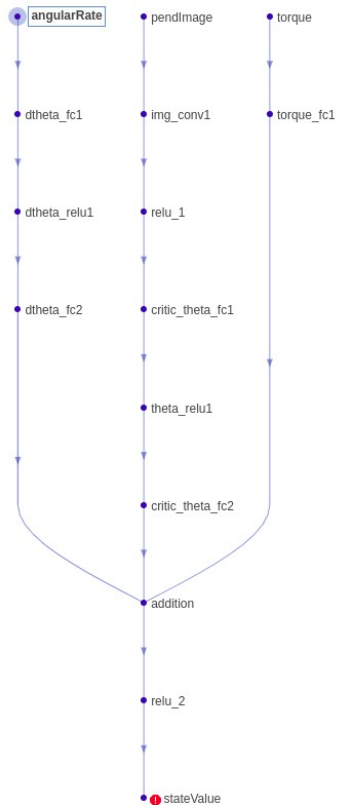
层

0

警告

2

错误



问题

发现于	消息
stateValue	未连接的输出。每个层输出必须连接到另一层的输入。
网络	缺少输出层。网络必须有至少一个输出层。

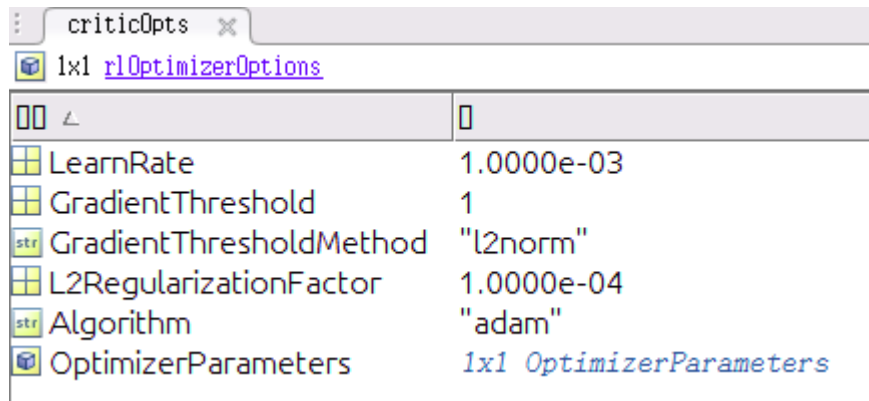
分析结果

名称	类型	激活	可学习参数属性	状态
1 angularRate 1×1×1 图像	图像输入	1(S) × 1(S) × 1(C) × 1(B)	-	-
2 dtheta_fc1 400 全连接层	全连接	1(S) × 1(S) × 400(C) × 1(B)	Weights 400 × 1 Bias 400 × 1	-
3 dtheta_relu1 ReLU	ReLU	1(S) × 1(S) × 400(C) × 1(B)	-	-
4 dtheta_fc2 300 全连接层	全连接	1(S) × 1(S) × 300(C) × 1(B)	Weights 300 × 4... Bias 300 × 1	-
5 torque 1×1×1 图像	图像输入	1(S) × 1(S) × 1(C) × 1(B)	-	-
6 torque_fc1 300 全连接层	全连接	1(S) × 1(S) × 300(C) × 1(B)	Weights 300 × 1 Bias 300 × 1	-
7 pendImage 50×50×1 图像	图像输入	50(S) × 50(S) × 1(C) × 1(B)	-	-
8 img_conv1 2 10×10 卷积: 步幅 [5 5], 填充 's...	二维卷积	10(S) × 10(S) × 2(C) × 1(B)	Weights 10 × 10 × ... Bias 1 × 1 × 2	-
9 relu_1 ReLU	ReLU	10(S) × 10(S) × 2(C) × 1(B)	-	-
10 critic_theta_fc1 400 全连接层	全连接	1(S) × 1(S) × 400(C) × 1(B)	Weights 400 × 2... Bias 400 × 1	-
11 theta_relu1 ReLU	ReLU	1(S) × 1(S) × 400(C) × 1(B)	-	-
12 critic_theta_fc2 300 全连接层	全连接	1(S) × 1(S) × 300(C) × 1(B)	Weights 300 × 4... Bias 300 × 1	-
13 addition 加法: 将 3 个输入按元素相加	加法	1(S) × 1(S) × 300(C) × 1(B)	-	-
14 relu_2 ReLU	ReLU	1(S) × 1(S) × 300(C) × 1(B)	-	-
15 stateValue 1 全连接层	全连接	1(S) × 1(S) × 1(C) × 1(B)	Weights 1 × 300 Bias 1 × 1	-

Export Network from Deep Network Designer

Specify options for the critic representation using rlOptimizerOptions.

```
criticOpts = rlOptimizerOptions('LearnRate',1e-03,'GradientThreshold',1);
```



The screenshot shows the MATLAB Deep Network Designer interface. At the top, there is a tab labeled 'criticOpts'. Below the tab, the table structure is displayed as '1x1 rlOptimizerOptions'. The table itself has two columns: the first column contains property names with icons indicating their data type (numeric, string, or object), and the second column contains the corresponding values. The properties and values are: LearnRate (numeric, 1.0000e-03), GradientThreshold (numeric, 1), GradientThresholdMethod (string, 'l2norm'), L2RegularizationFactor (numeric, 1.0000e-04), Algorithm (string, 'adam'), and OptimizerParameters (object, 1x1 OptimizerParameters).

criticOpts	
1x1	rlOptimizerOptions
LearnRate	1.0000e-03
GradientThreshold	1
GradientThresholdMethod	"l2norm"
L2RegularizationFactor	1.0000e-04
Algorithm	"adam"
OptimizerParameters	1x1 OptimizerParameters

Export Network from Deep Network Designer

Create the critic representation using the specified deep neural network lgraph and options. You must also specify the action and observation info for the critic, which you obtain from the environment interface.

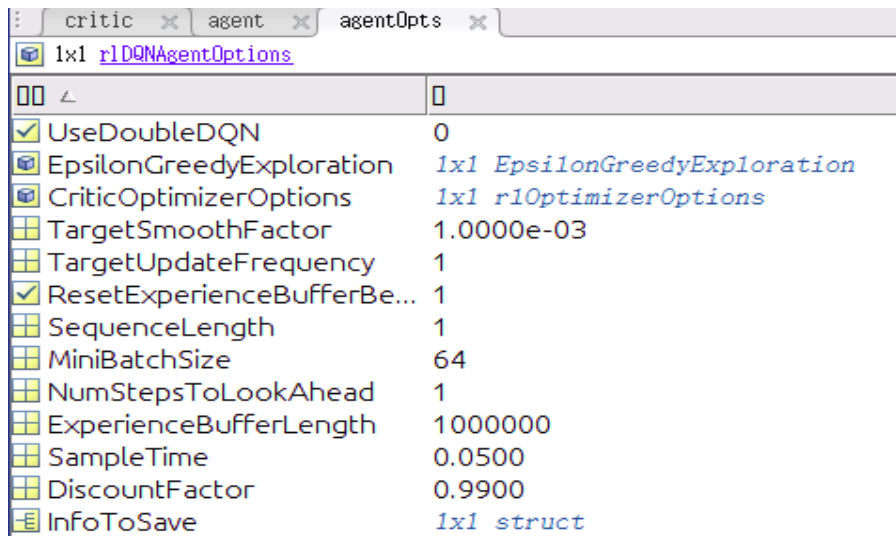
```
net = dlnetwork(lgraph);  
critic = rlQValueFunction(net,obsInfo,actInfo,...  
    "ObservationInputNames",  
    ["pendImage","angularRate"],"ActionInputNames","torque");
```

critic	
1x1 rlQValueFunction	
ObservationInfo	2x1 rlNumericSpec
ActionInfo	1x1 rlFiniteSetSpec
UseDevice	"cpu"

Export Network from Deep Network Designer

To create the DQN agent, first specify the DQN agent options using `rlDQNAgentOptions`.

```
agentOpts = rlDQNAgentOptions(...  
    'UseDoubleDQN',false,...  
    'CriticOptimizerOptions',criticOpts,...  
    'ExperienceBufferLength',1e6,...  
    'SampleTime',env.Ts);  
agentOpts.EpsilonGreedyExploration.EpsilonDecay = 1e-5;
```

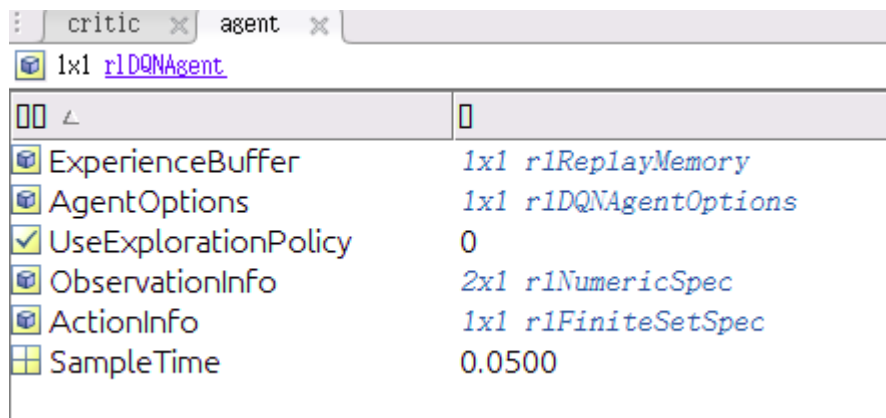


critic x agent x agentOpts x	
1x1 rlDQNAgentOptions	
<input checked="" type="checkbox"/> UseDoubleDQN	0
EpsilonGreedyExploration	1x1 EpsilonGreedyExploration
CriticOptimizerOptions	1x1 rlOptimizerOptions
TargetSmoothFactor	1.0000e-03
TargetUpdateFrequency	1
<input checked="" type="checkbox"/> ResetExperienceBufferBe...	1
SequenceLength	1
MiniBatchSize	64
NumStepsToLookAhead	1
ExperienceBufferLength	1000000
SampleTime	0.0500
DiscountFactor	0.9900
InfoToSave	1x1 struct

Export Network from Deep Network Designer

Then, create the DQN agent using the specified critic representation and agent options.

```
agent = rlDQNAgent(critic,agentOpts);
```



critic x agent x	
1x1 rlDQNAgent	
ExperienceBuffer	1x1 rlReplayMemory
AgentOptions	1x1 rlDQNAgentOptions
UseExplorationPolicy	0
ObservationInfo	2x1 rlNumericSpec
ActionInfo	1x1 rlFiniteSetSpec
SampleTime	0.0500

Train Agent

To train the agent, first specify the training options. For this example, use the following options.

- Run each training for at most 5000 episodes, with each episode lasting at most 500 time steps.
- Display the training progress in the Episode Manager dialog box (set the Plots option) and disable the command line display (set the Verbose option to false).
- Stop training when the agent receives an average cumulative reward greater than -1000 over the default window length of five consecutive episodes. At this point, the agent can quickly balance the pendulum in the upright position using minimal control effort.

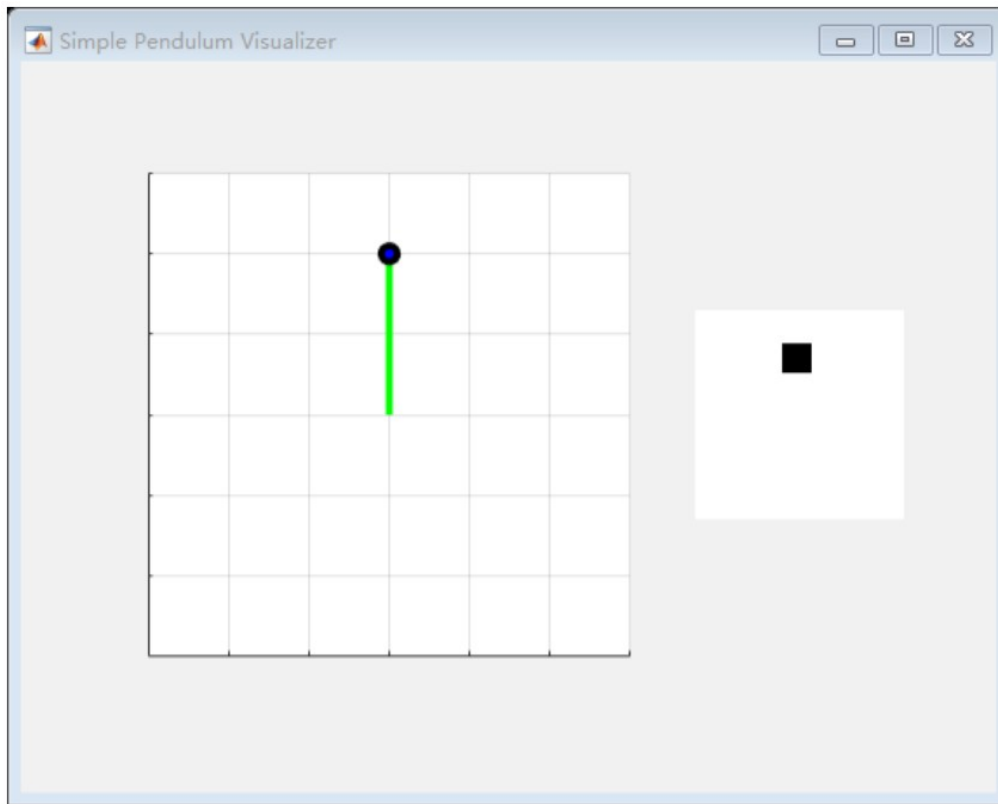
Train Agent

```
trainOpts = rlTrainingOptions(...  
    'MaxEpisodes',5000,...  
    'MaxStepsPerEpisode',500,...  
    'Verbose',false,...  
    'Plots','training-progress',...  
    'StopTrainingCriteria','AverageReward',...  
    'StopTrainingValue',-1000);
```

Train Agent

You can visualize the pendulum system during training or simulation by using the plot function.

`plot(env)`



Train Agent

Train the agent using the train function. This is a computationally intensive process that takes several hours to complete. To save time while running this example, load a pretrained agent by setting doTraining to false. To train the agent yourself, set doTraining to true.

```
doTraining = false;
```

```
if doTraining
```

```
    % Train the agent.
```

```
    trainingStats = train(agent,env,trainOpts);
```

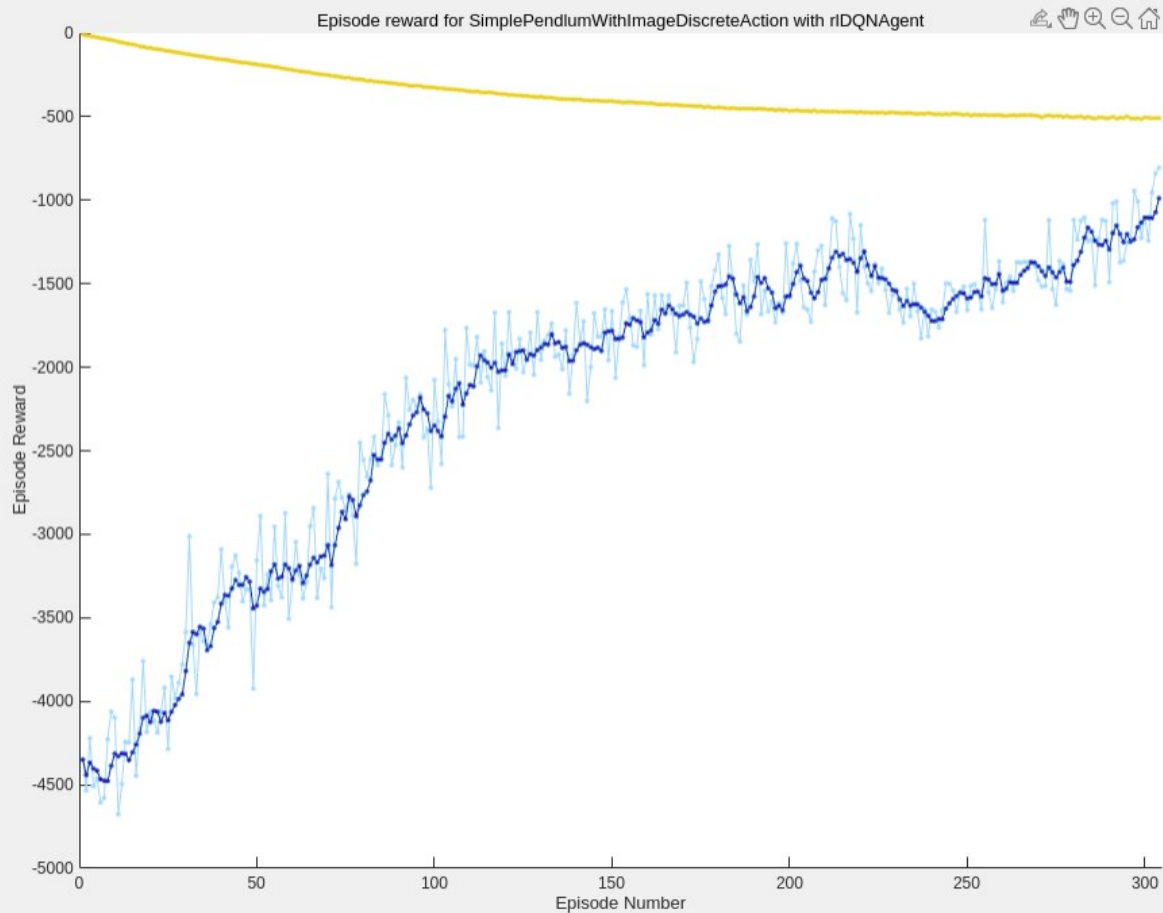
```
else
```

```
    % Load pretrained agent for the example.
```

```
    load('MATLABPendImageDQN.mat','agent');
```

```
end
```

rIDQNAgent



Training Progress

Training stopped

Episode number: 304/5000
Start time: 2022-12-02 21:03:11
Duration: 10:40:14
Final result: Training finished after all agents reached stop training criteria.

Training Information

Agent	Status
rIDQNAgent	Training finished

Episode Information (rIDQNAgent):

Episode reward: -804.8288
Average reward: -988.555
Episode Q0: -508.8342

[More Details...](#)

Plot Options

☒ Show EpisodeQ0☐ Show last N episodes

Episode reward Episode Q0
Average reward

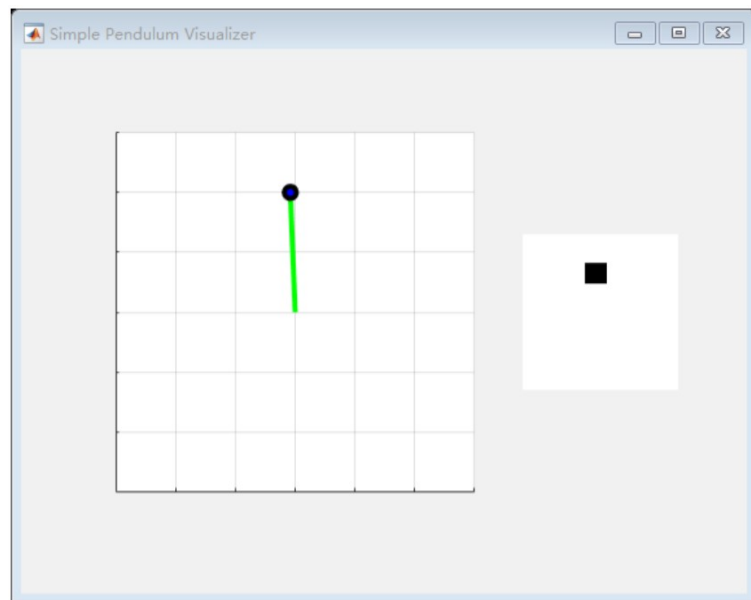
Simulate DQN Agent

To validate the performance of the trained agent, simulate it within the pendulum environment. For more information on agent simulation, see `rlSimulationOptions` and `sim`.

```
simOptions = rlSimulationOptions('MaxSteps',500);  
experience = sim(env,agent,simOptions);  
totalReward = sum(experience.Reward)
```

```
totalReward = -888.9802
```

experience	
1x1 struct 包含 5 个字段	
Observation	1x1 struct
Action	1x1 struct
Reward	1x1 double timeseries
IsDone	1x1 uint8 timeseries
SimulationInfo	1x1 struct



DRL 示例代码

% Create Agent Using Deep Network Designer and Train Using Image Observations

openExample('control_deeplearning/CreateAgentUsingDNDAndTrainUsingImageObsExample')