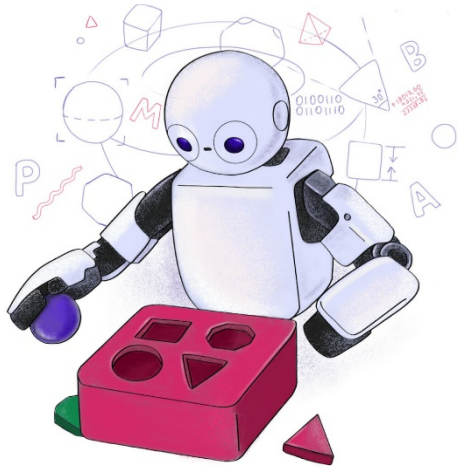
TP558 - Tópicos avançados em Machine Learning:

Algoritmo DDPG

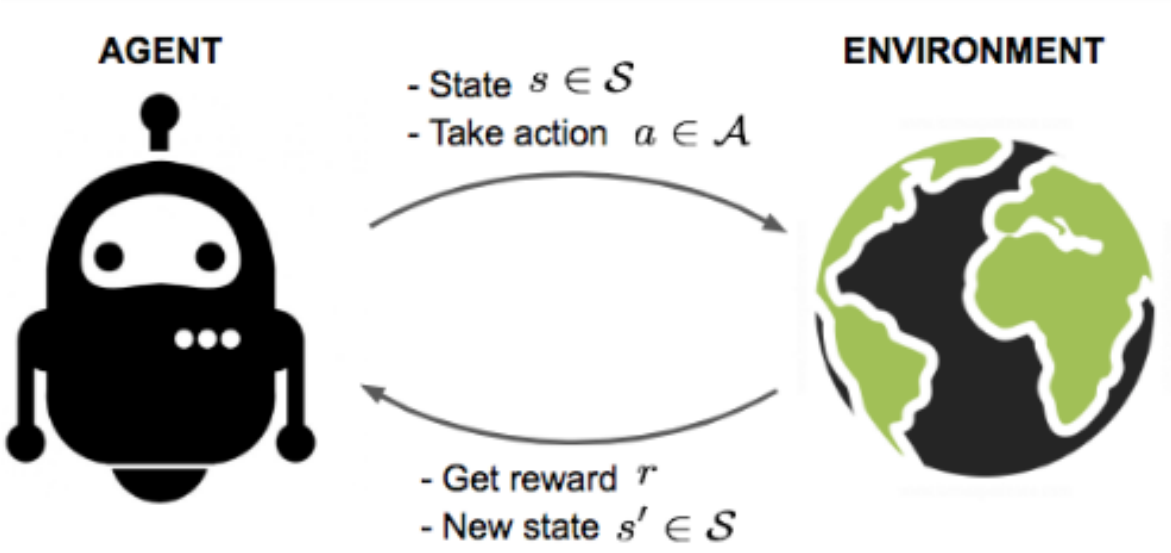


Inatel

Felipe Batista Faro Pinto
felipe.pinto@dtel.inatel.br

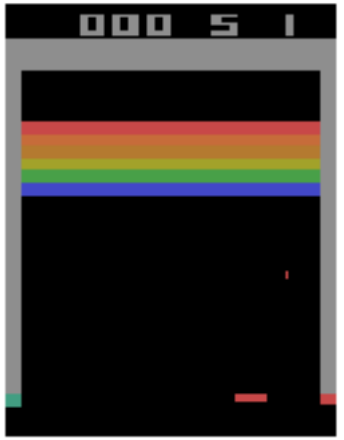
Introdução

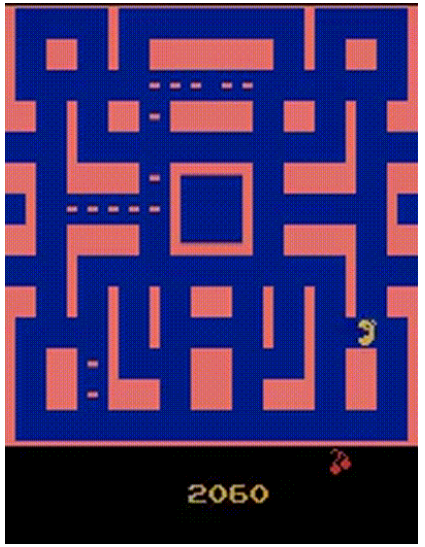
- Resolução de tarefas complexas a partir de dados de sensoriamento
- O aprendizado por reforço se torna interessante devido ao aprendizado constante sem a necessidade de um dataset
- Em 2015 houve a criação da Deep Q-Network (DQN), que combina aprendizado profundo com aprendizado por reforço



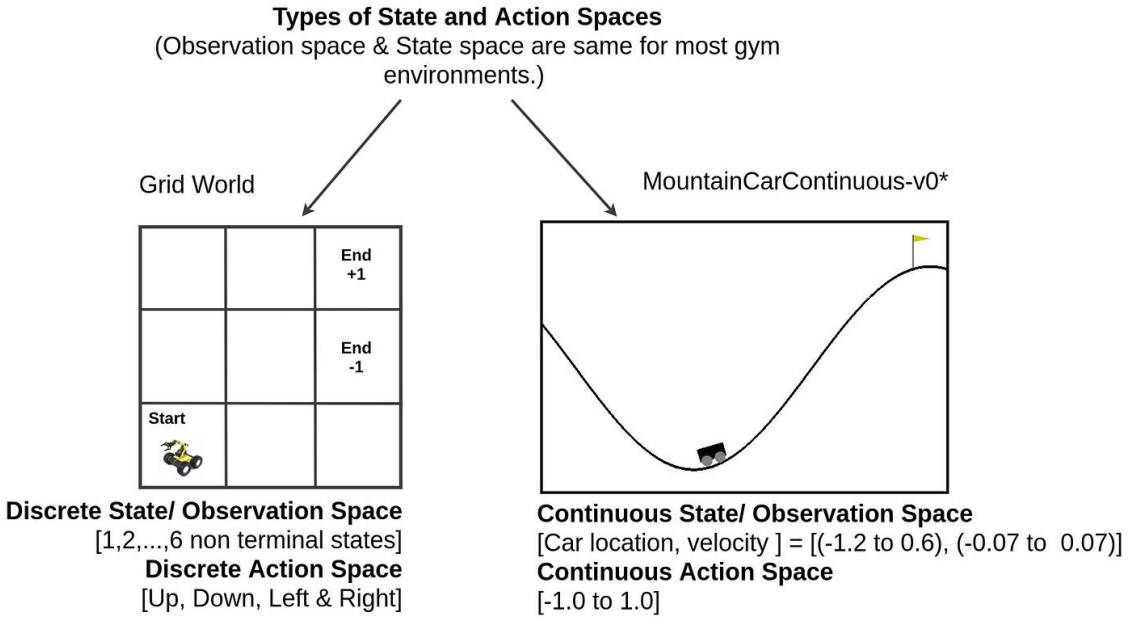
Aprendizado por reforço

Introdução





- DQN analisa problemas discretos
- Muitos problemas cotidianos possui espaço de ações continuas
- DQN se torna impraticável em espaço de ações continuas



Introdução

CONTINUOUS CONTROL WITH DEEP REINFORCEMENT LEARNING

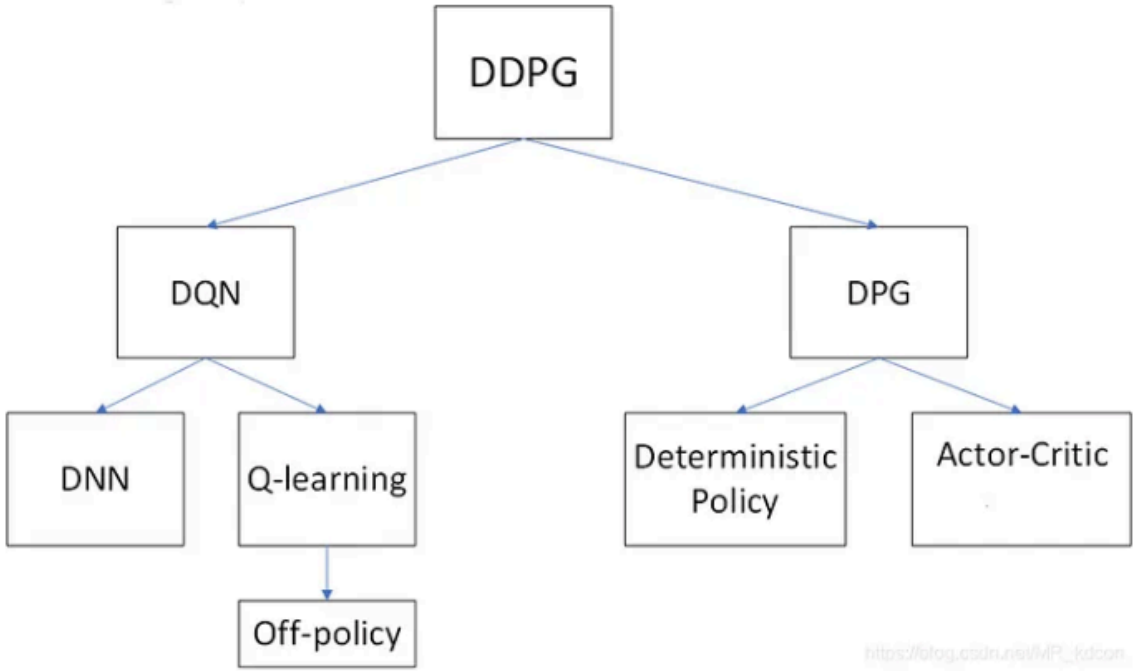
Timothy P. Lillicrap*, Jonathan J. Hunt*, Alexander Pritzel, Nicolas Heess,
Tom Erez, Yuval Tassa, David Silver & Daan Wierstra
Google Deepmind
London, UK

2016

- **DDPG (*deep deterministic policy gradient*)**
- Lida com problemas de ações contínuos
- Baseado no DPG e DQN
- Combina DPG com redes neurais profundas, permitindo aprendizado em espaços de ações contínuas de alta dimensão
- Pode ser aplicado em observações de baixas dimensões, como coordenadas cartesianas, e também aprender com imagens de entrada (pixels)
- Aplicações em robôs ou veículos autônomos, vídeo games e simuladores, otimização de processos industriais

Introdução

- O **DDPG** aprende a partir de amostras passadas, independente da política atual (*Off-policy*)
- O **DDPG** é *Model-free*, ou seja ele não modela o ambiente, e sim aprende uma determinada politica e tenta tomar a melhor ação



Cartpole, reaching task, Grasp and Move Task, A Puck-Hitting Task, A Monoped Balancing Task, Two Locomotion Tasks, Torcs (Driving Simulator)

Fundamentação teórica

O DDPG apresenta o ator e o crítico:

Ator:

- O ator é responsável por determinar a política da ação, ou seja, dado um estado atual, ele decide qual ação deve ser tomada. No DDPG, o ator é uma rede neural que mapeia os estados observados do ambiente para ações contínuas.

Crítico:

- O crítico avalia a qualidade das ações selecionadas pelo ator. Ele estima a função de valor de ação $Q(s,a)$, que representa o valor esperado de um estado s ao executar uma ação a.
- No DDPG, o crítico é outra rede neural que recebe como entrada tanto o estado quanto a ação e estima o valor Q. Essa estimativa é usada para calcular quão boa é a ação tomada pelo ator.

Fundamentação teórica

Policy gradient

- Uma política é uma estratégia que um agente usa em busca de objetivos. A política dita as ações que o agente toma como função do estado do agente e do ambiente.
- Cada vez que o agente interage com o ambiente, ajustamos os parâmetros θ (pesos) da rede neural para que as ações "boas" sejam amostradas com mais probabilidade no futuro. Repetimos esse processo até que a rede de políticas convirja para a política ótima.
- O objetivo do *policy gradient* é maximizar as recompensas futuras esperadas totais, ajustando iterativamente os parâmetros θ da nossa rede de políticas para maximizar a recompensa.

Fundamentação teórica

Aprendizado *Off-policy*

- A aprendizagem *off-policy* refere-se a uma abordagem de aprendizagem por reforço em que o algoritmo de aprendizagem pode aprender com os dados gerados por uma política diferente daquela que está sendo otimizada atualmente.
- Isso permite maior flexibilidade na forma como os dados são coletados e usados para treinamento.
- Off-policy permite que o agente explore continuamente porque pode usar uma política de exploração diferente enquanto aprende a política ideal.

Fundamentação teórica

Politica determinística e estocástica

O DDPG aplica política determinística

- **Determinística:** Denotada por μ, mapeia cada estado s diretamente a uma ação específica a. Isso significa que para um dado estado s, a ação a escolhida pela política é sempre a mesma.

$$a = \mu(s)$$

- Políticas determinísticas são especialmente úteis em espaços de ação contínuos, onde o espaço de ações é infinito, tornando difícil a aplicação de métodos baseados em políticas estocásticas, que exigem a definição de uma distribuição de probabilidade sobre o espaço de ações.

Fundamentação teórica

Politica determinística e estocástica

- **Estocástica:** Denotada por π, define uma distribuição de probabilidade sobre as ações para cada estado s. Isso significa que, dado um estado s, a ação a é escolhida com uma certa probabilidade.

$$a \approx \pi(a|s)$$

- Políticas estocásticas são frequentemente usadas em espaços de ação discretos, onde é viável definir uma distribuição de probabilidade sobre um conjunto finito de ações.

Fundamentação teórica

Exploração:

- Ação determinística do ator com a adição de um ruído durante o processo de treinamento
- Usualmente utiliza-se o processo de ruído **Ornstein-Uhlenbeck**
- Usado para gerar exploração temporalmente correlacionada para melhorar a eficiência de exploração
- Esse ruído é adicionado às ações sugeridas pela política determinística para garantir que o agente explore adequadamente o espaço de ação e evite ficar presos em políticas sub-ótimas

Fundamentação teórica

Desafios no aprendizado por reforço:

- Um desafio ao usar redes neurais para aprendizagem por reforço é que a maioria dos algoritmos de otimização assume que as amostras são distribuídas de forma independente e idêntica.
- Obviamente, quando as amostras são geradas a partir da exploração sequencial de um ambiente, essa suposição não é mais válida.
- A implementação direta do aprendizado Q com redes neurais provou ser instável em muitos ambientes. Como a rede Q(s,a) que está sendo atualizada também é usada no cálculo do valor alvo, a atualização Q está sujeita a divergências.

Fundamentação teórica

Implementações importantes para solucionar os problemas:

- ***Replay buffer:*** Memória para garantir transições descorrelacionadas durante o treinamento.
- ***Target network:*** Redes separadas que são atualizadas mais lentamente para adicionar estabilidade ao treinamento.
- ***Batch normalization*****:** Técnica para normalizar as características das amostras em um *minibatch* para ter média e variância unitária.

Fundamentação teórica

***Replay buffer*:**

- É uma memória cache de tamanho finito.
- Quando o buffer de repetição está cheio, as amostras mais antigas são descartadas.
- Em cada passo de tempo, o ator e o crítico são atualizados amostrando um *minibatch* do buffer.
- Permite que algoritmo se beneficie do aprendizado através de um conjunto de transições não correlacionadas.
- Essa implementação garante uma maior eficiência no aprendizado

Fundamentação teórica

***Target networks* (redes alvo):**

- As redes alvo são cópias com atraso temporal de suas redes originais que rastreiam lentamente as redes aprendidas. Usar essas redes melhora significativamente a estabilidade no aprendizado.
- Uma das principais funções das "*target networks*" é fornecer alvos de aprendizado mais estáveis e menos voláteis para a rede do crítico durante o treinamento do DDPG. Isso ajuda a estabilizar o treinamento e melhorar a convergência do algoritmo.
- As redes-alvo são atualizadas lentamente com base nas redes principais utilizando um fator de atualização tau, que é muito menor que 1, para garantir que as mudanças sejam pequenas e incrementais. Essa técnica é essencial para evitar grandes oscilações nos valores das redes-alvo, proporcionando um treinamento mais estável.

Fundamentação teórica

Batch normalization:

- Diferentes componentes do observação pode ter diferentes unidades físicas (por exemplo, posições versus velocidades) e os intervalos podem variar entre ambientes. Isso pode dificultar o aprendizado eficaz da rede e pode dificultar a achar hiperparâmetros que generalizem entre ambientes com diferentes escalas de valores de estado.

- Uma abordagem para esse problema é dimensionar manualmente as características para que fiquem em intervalos semelhantes entre ambientes e unidades. Essa técnica é chamada *batch normalization,* e consiste em normalizar cada dimensão nas amostras em um minilote para ter média e variância unitária.

- Com a normalização em lote, pode-se aprender de forma eficaz diversas tarefas com diferentes tipos de unidades, sem a necessidade de garantir manualmente que as unidades estavam dentro de um intervalo definido.

Fundamentação teórica

Algorithm 1 DDPG algorithm

Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ.
Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer R
for episode = 1, M **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial observation state s_1
  for t = 1, T **do**
    Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
    Execute action a_t and observe reward r_t and observe new state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in R
    Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R
    Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
    Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
    Update the actor policy using the sampled policy gradient:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N} \sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_t, a=\mu(s_t)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s_t}$$

    Update the target networks:

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$
$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$

  end for
end for

Arquitetura

```python
def get_actor():
    last_init = keras.initializers.RandomUniform(minval=-0.003, maxval=0.003)

    inputs = layers.Input(shape=(num_states,))
    out = layers.Dense(256, activation="relu")(inputs)
    out = layers.Dense(256, activation="relu")(out)
    outputs = layers.Dense(1, activation="tanh", kernel_initializer=last_init)(out)

    outputs = outputs * upper_bound
    model = keras.Model(irputs, outputs)
    return model
```

Rede neural do ator

- **Entrada**: Estado do ambiente.
- **Saídas**: Ação contínua, escalada pelo limite superior do espaço de ação.
- **Ativações**: Funções relu para camadas ocultas e tanh para a camada de saída

```python
def get_critic():
    state_input = layers.Input(shape=(num_states,))
    state_out = layers.Dense(16, activation="relu")(state_input)
    state_out = layers.Dense(32, activation="relu")(state_out)

    action_input = layers.Input(shape=(num_actions,))
    action_out = layers.Dense(32, activation="relu")(action_input)

    concat = layers.Concatenate()([state_out, action_out])

    out = layers.Dense(256, activation="relu")(concat)
    out = layers.Dense(256, activation="relu")(out)
    outputs = layers.Dense(1)(out)

    model = keras.Model([state_input, action_input], outputs)

    return model
```

Rede neural do crítico

- **Entradas**: Estado e ação.
- **Saída**: Valor Q, que avalia a qualidade da ação dada o estado.
- **Ativações**: Funções relu para todas as camadas ocultas.

Treinamento e otimização

```
with tf.GradientTape() as tape:
    actions = actor_model(state_batch, training=True)
    critic_value = critic_model([state_batch, actions], training=True)
    actor_loss = -keras.ops.mean(critic_value)

actor_grad = tape.gradient(actor_loss, actor_model.trainable_variables)
actor_optimizer.apply_gradients(
    zip(actor_grad, actor_model.trainable_variables)
)
```

Função de Erro do Ator

•**Erro**: Negativo da média dos valores Q preditos pela rede crítica para as ações sugeridas pela rede do ator (maximização da recompensa).
•**Otimizador**: Adam, com taxa de aprendizado especificada.

```
with tf.GradientTape() as tape:
    target_actions = target_actor(next_state_batch, training=True)
    y = reward_batch + gamma * target_critic(
        [next_state_batch, target_actions], training=True
    )
    critic_value = critic_model([state_batch, action_batch], training=True)
    critic_loss = keras.ops.mean(keras.ops.square(y - critic_value))

critic_grad = tape.gradient(critic_loss, critic_model.trainable_variables)
critic_optimizer.apply_gradients(
    zip(critic_grad, critic_model.trainable_variables)
)
```

Função de Erro do Crítico

•**Erro**: Mean Squared Error (MSE) entre o valor alvo y e o valor Q predito pela rede crítica.
•**Otimizador**: Adam, com taxa de aprendizado especificada.

Vantagens

1. **Apropriado para Espaços de Ação Contínuos:**
   O DDPG é especialmente adequado para problemas com espaços de ação contínuos, onde as ações podem ser uma variedade infinita de valores. Isso permite que o algoritmo seja aplicado em uma ampla gama de problemas do mundo real, como robótica e controle de motores.
2. **Aprendizado Off-Policy:**
   O DDPG utiliza aprendizado off-policy, o que significa que o agente pode aprender com experiências passadas armazenadas na memória de repetição. Isso aumenta a eficiência do aprendizado e permite uma melhor reutilização dos dados de experiência.
3. **Estabilidade do Treinamento:**
   O uso de "target networks" e “replay buffer”, junto com a política *off-policy* contribui para a estabilidade do treinamento. Isso ajuda a reduzir oscilações e a melhorar a convergência do algoritmo, tornando-o mais robusto e confiável.
4. **Robustez em ambientes ruidosos:**
   O ruído controlado do DDPG ajuda o agente a explorar diferentes ações mesmo em ambientes ruidosos, o que é crucial para encontrar políticas ótimas em ambientes onde o comportamento do sistema pode ser incerto ou variável. Isso o torna adequado para aplicações onde a precisão é crucial, como navegação autônoma.

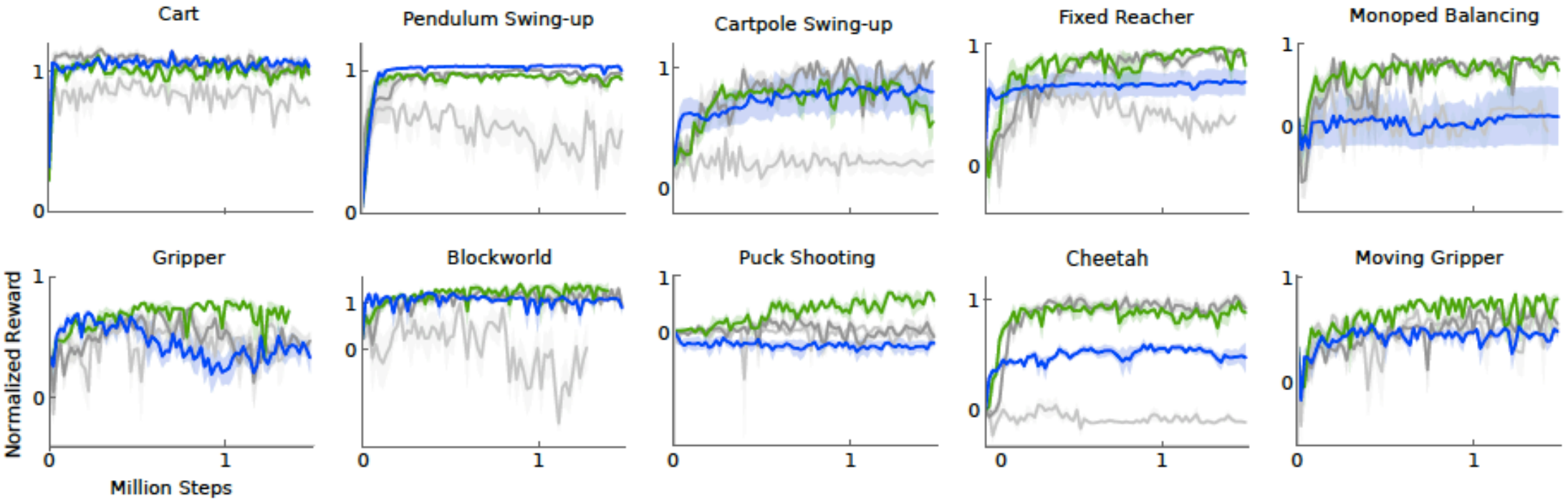
Desvantagens

1. **Sensibilidade a Hiperparâmetros:**
O DDPG é altamente sensível aos hiperparâmetros, como a taxa de aprendizado, o fator de desconto, o fator de atualização das redes-alvo, e os parâmetros da rede neural (número de camadas, unidades por camada, etc.).

2. **Falta de Exploração Eficiente:**
DDPG utiliza uma política determinística, o que pode levar a problemas de exploração inadequada. Embora ruídos (como Ornstein-Uhlenbeck) sejam adicionados às ações para incentivar a exploração, isso pode não ser suficiente em ambientes complexos ou com alta dimensionalidade.

3. **Instabilidade e Divergência:**
DDPG, como outros métodos baseados em gradiente profundo, pode sofrer de instabilidades durante o treinamento. A função de valor e a política são representadas por redes neurais profundas que são treinadas simultaneamente, o que pode levar a problemas de divergência. Para isso é necessário implementar corretamente as redes alvos e replay buffer.

Comparação com outros algoritmos



Cinza claro : DPG original com *batch normalization*
Cinza escuro: DPG original com *target networks*
Verde: DPG original com *target networks* e *batch normalization*
Azul: DPG original com *target networks* com entradas a partir de pixels

***Target networks* são cruciais!**

Comparação com outros algoritmos

Tabela de performance do DDPG em comparação com o DPG (com replay buffer e batch normalization), para diversos ambientes.

- 2.5 milhões de etapas de treinamento
- 5 execuções (média e maior valor)
- Scores normalizados, exceto o torcs
- “lowd” são entradas low-dimensional
- “pix” são entradas de pixel (high-dimensional)

environment	$R_{av,lowd}$	$R_{best,lowd}$	$R_{av,pix}$	$R_{best,pix}$	$R_{av,cntrl}$	$R_{best,cntrl}$
blockworld1	1.156	1.511	0.466	1.299	-0.080	1.260
blockworld3da	0.340	0.705	0.889	2.225	-0.139	0.658
canada	0.303	1.735	0.176	0.688	0.125	1.157
canada2d	0.400	0.978	-0.285	0.119	-0.045	0.701
cart	0.938	1.336	1.096	1.258	0.343	1.216
cartpole	0.844	1.115	0.482	1.138	0.244	0.755
cartpoleBalance	0.951	1.000	0.335	0.996	-0.468	0.528
cartpoleParallelDouble	0.549	0.900	0.188	0.323	0.197	0.572
cartpoleSerialDouble	0.272	0.719	0.195	0.642	0.143	0.701
cartpoleSerialTriple	0.736	0.946	0.412	0.427	0.583	0.942
cheetah	0.903	1.206	0.457	0.792	-0.008	0.425
fixedReacher	0.849	1.021	0.693	0.981	0.259	0.927
fixedReacherDouble	0.924	0.996	0.872	0.943	0.290	0.995
fixedReacherSingle	0.954	1.000	0.827	0.995	0.620	0.999
gripper	0.655	0.972	0.406	0.790	0.461	0.816
gripperRandom	0.618	0.937	0.082	0.791	0.557	0.808
hardCheetah	1.311	1.990	1.204	1.431	-0.031	1.411
hopper	0.676	0.936	0.112	0.924	0.078	0.917
hyq	0.416	0.722	0.234	0.672	0.198	0.618
movingGripper	0.474	0.936	0.480	0.644	0.416	0.805
pendulum	0.946	1.021	0.663	1.055	0.099	0.951
reacher	0.720	0.987	0.194	0.878	0.231	0.953
reacher3daFixedTarget	0.585	0.943	0.453	0.922	0.204	0.631
reacher3daRandomTarget	0.467	0.739	0.374	0.735	-0.046	0.158
reacherSingle	0.981	1.102	1.000	1.083	1.010	1.083
walker2d	0.705	1.573	0.944	1.476	0.393	1.397
torcs	-393.385	1840.036	-401.911	1876.284	-911.034	1961.600

Aplicações DDPG

Carrinho evitando obstáculo: https://www.youtube.com/watch?v=y3cLHBNkFKo&ab_channel=MaxFerguson

Gym mujoco: https://www.youtube.com/watch?v=iFg5lcUzSYU&ab_channel=NiloFreitas

Bipedal walker: https://www.youtube.com/watch?v=_uOijzF4V6U&ab_channel=JosephLowman

Exemplo(s) de aplicação

- Este exemplo demonstra a aplicabilidade do DDPG para otimização do pêndulo invertido, de modo a tentar estabilizar o mesmo na vertical.
- Link para o colab: https://colab.research.google.com/drive/1xeYSWCLI75FY_sKBRdhh-NnWdPAYyW2-?usp=sharing

Quiz

- Link para acesso ao quiz: https://forms.gle/Ru4ZaqwsfrmixMdd8

Referências

- Lillicrap, Timothy P., et al. "Continuous control with deep reinforcement learning." arXiv preprint arXiv:1509.02971 (2015).
- Keras. Deep Deterministic Policy Gradients (DDPG) - Pendulum Example. Disponível em: https://keras.io/examples/rl/ddpg_pendulum/. 2024.
- Medium. Deep Deterministic Policy Gradients Explained. Disponível em: https://towardsdatascience.com/deep-deterministic-policy-gradients-explained-2d94655a9b7b. 2019.
- Medium. How DDPG (Deep Deterministic Policy Gradient) Algorithms Works in Reinforcement Learning. Disponível em: https://medium.com/@amaresh.dm/how-ddpg-deep-deterministic-policy-gradient-algorithms-works-in-reinforcement-learning-117e6a932e68. 2022.
- Medium. Deep Deterministic Policy Gradient (DDPG): An Off-Policy Reinforcement Learning Algorithm. Disponível em: https://medium.com/intro-to-artificial-intelligence/deep-deterministic-policy-gradient-ddpg-an-off-policy-reinforcement-learning-algorithm-38ca8698131b. 2020.

Obrigado!