

Mountain Car & Half Cheetah

Pelinsu Sağlam
Esma Hatun Akbulut
Nisa Büyüktas
Ayçanur Güç





İÇERİK

Proximal Policy Optimization (PPO)

PPO Sözde Kodu ve UML Diyagramı

Soft Actor-Critic (SAC)

SAC Sözde Kodu ve UML Diyagramı

Deep Deterministic Policy Gradient (DDPG)

DDPG Sözde Kodu ve UML Diyagramı

Q-Değeri Hesaplama Farkları

Mountain Car Problemi

Seçilen Algoritma ve Problem Hakkında Yapılan Çalışmalar

Mountain Car mantığı nedir?

Mountain Car Sınırları Nedir?

Mountain Car Ayrık mı Sürekli mi?

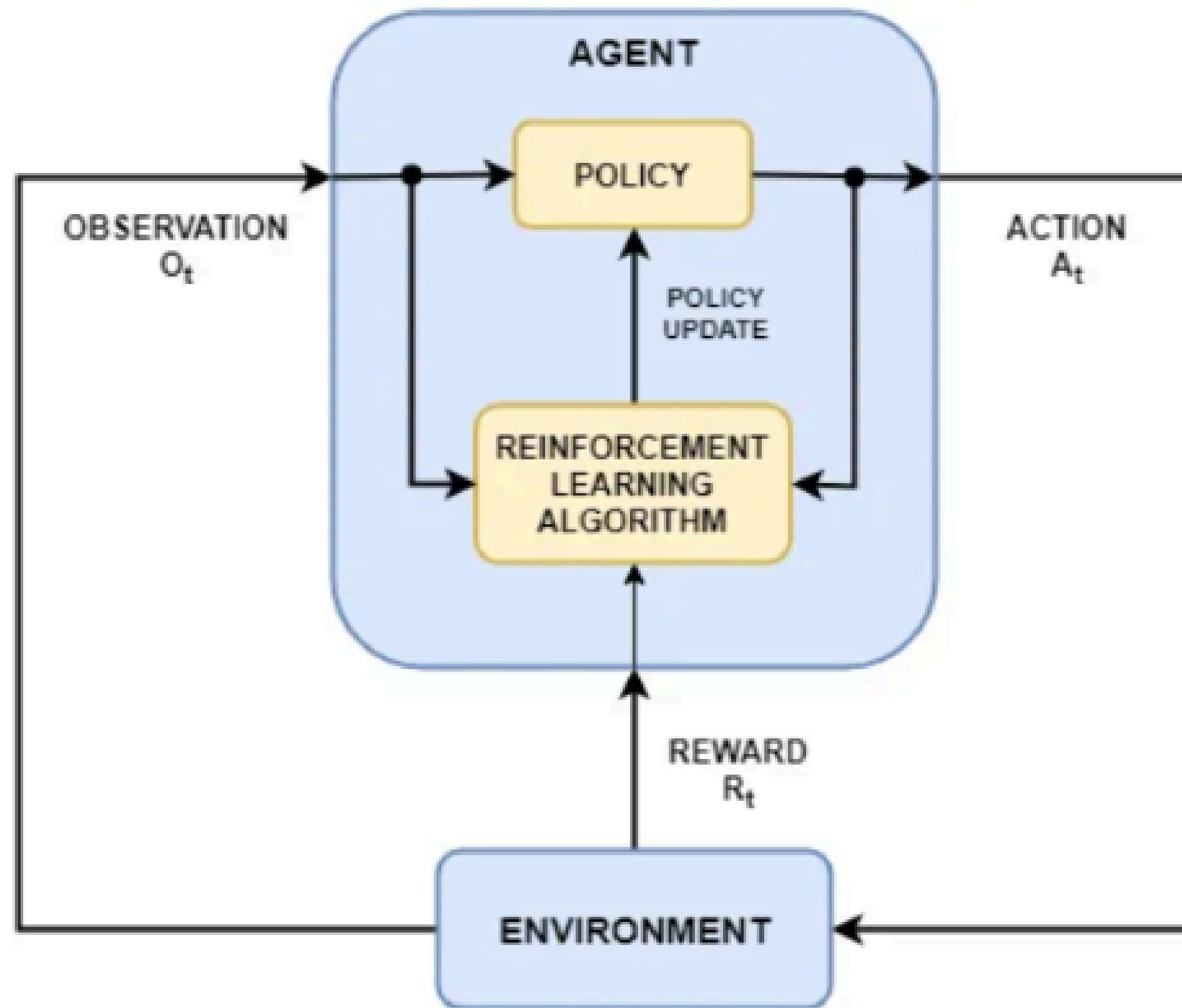
Half Cheetah Problemi

Seçilen Algoritma ve Problem Hakkında Yapılan Çalışmaları

Half Cheetah mantığı nedir?

Half Cheetah Sınırları Nedir?

Half Cheetah Ayrık mı Sürekli mi?



A general representation of reinforcement learning one-time step process. Source: [1]

Proximal Policy Optimization (PPO)

PPO, yüksek doğrulukta politika güncellemeleri yaparken dengeleyici bir optimizasyon sağlar. PPO, politika güncellemelerini sınırlamak için "clip" fonksiyonu gibi mekanizmalar kullanır, bu da ani politika değişikliklerini önleyerek öğrenmeyi kararlı hale getirir. PPO algoritması, hem yüksek başarı oranı hem de veri etkinliği nedeniyle sürekli ve ayrık eylem alanlarında yaygın olarak kullanılır.

Proximal Policy Optimization (PPO)

PPO algoritması, politika temelli olduğu için doğrudan Q-değerini hesaplamaz; bunun yerine avantaj fonksiyonunu kullanır. Avantaj fonksiyonu, bir aksiyonun ne kadar iyi olduğunu gösterir ve şu şekilde hesaplanır:

Burada:

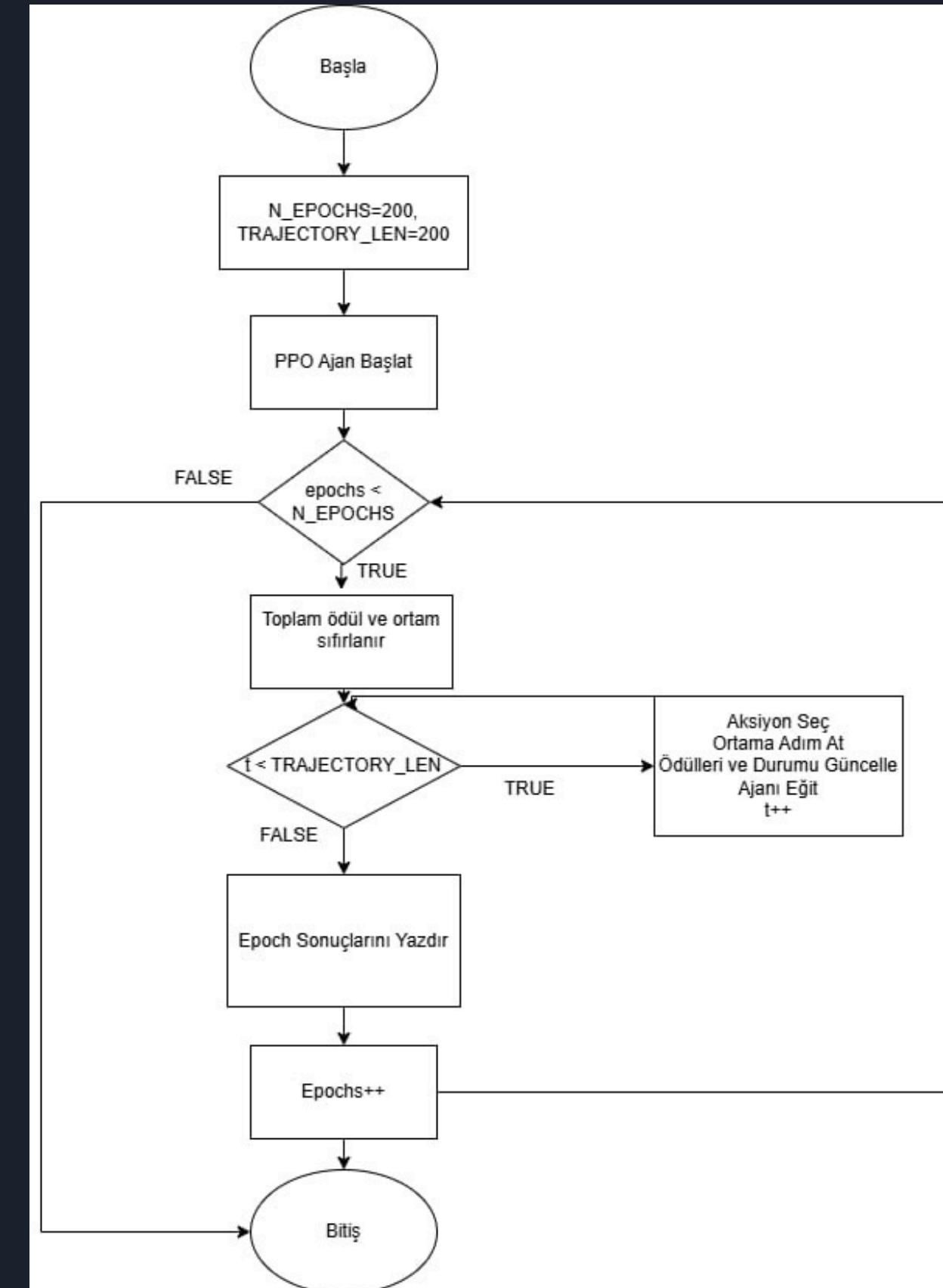
- $Q(s,a)$: Belirli bir durumda belirli bir aksiyonun beklenen ödülü.
- $V(s)$: Sadece durumun beklenen ödülü.

PPO'nun Farkı:

- PPO doğrudan Q-değerini optimize etmez, bunun yerine avantaj fonksiyonuna odaklanır.
- Avantaj fonksiyonu ile politikayı optimize ederek, politikanın fazla değişimini engelleyen bir mekanizma kullanır.

PPO Sözde Kodu ve UML Diyagramı

BASLA
N_EPOCHS=200, TRAJECTORY_LEN=200
AJAN BASLAT, epochs=0
Döngü(epochs<N_EPOCHS) DO:
 toplam_ödül=0, mevcut durum=ORTAMI SIFIRLA, t=0
 Döngü(t<TRAJECTORY_LEN)DO:
 Aksion seç(mevcut durum,sac_agent)
 sonraki_durum,ödül,bitiş=ORTAM.ADIM_AT(aksiyon)
 toplam_ödül=toplam_ödül+ödül
 AJANIEĞİT(mevcut_durum,aksiyon,ödül,sonraki_durum,bitiş)#
 SAC ajanı eğit
 mevcut_durum=sonraki_durum
 EGER(bitiş==DOĞRU) İSE:
 Döngüden Çık
 t=t+1
 Eğitim Sonuçları Yazdır(epochs,toplam_ödül)
 epochs=epochs+1
BITİR



Soft Actor-Critic (SAC)

SAC, entropi düzenlemeye yöntemini kullanarak hem öğrenmeyi hızlandırır hem de politika keşfini teşvik eder. Sürekli eylem alanlarında, SAC'nin entropi bazlı ödüllendirme yaklaşımı, ajanların risk almasını teşvik ederken görev başarısını optimize eder. SAC, özellikle yüksek doğruluk gerektiren robotik görevlerde etkili olduğu için popüler hale gelmiştir.

Soft Actor-Critic (SAC)

SAC algoritmasında, Q-değeri, ajan için ödül ve bir sonraki durum hakkında bilgi sağlayan temel bir değerdir. SAC'de Q-değeri, Bellman denklemine dayalı olarak iki ayrı Q-ağı tarafından hesaplanır:

Bu denklemin parçaları:

- $r(s,a)$: Aksiyon sonrası elde edilen anlık ödül.
- γ : Gelecekteki ödüllerin bugünkü değere indirgeyen çarpan (indirim oranı).
- $V(s')$: Bir sonraki durumun beklenen ödülüdür.

Burada SAC, Q-değerlerini tahmin etmek için iki ayrı ağ kullanır ve bu ağların tahminlerinin ortalamasını alarak hatayı minimize eder. Entropiyi maksimize eden bir bileşen ekleyerek keşfi artırır, ancak Q-değeri hesaplamasında bu bileşen doğrudan yer almaz.

SAC Sözde Kodu ve UML Diyagramı

```
N_EPOCHS = 200
TRAJECTORY_LEN = 200
ORTAMI BAŞLAT
SAC AJANI BAŞLAT
epochs = 0
```

DÖNGÜ (epochs < N_EPOCHS) DO:

```
    toplam_ödül = 0
    mevcut_durum = ORTAMI SIFIRLA
    t = 0
```

DÖNGÜ (t < TRAJECTORY_LEN) DO:

```
    AKSİYON SEÇ (mevcut_durum, sac_agent) # SAC ajanı mevcut duruma göre aksiyon seçer
    sonraki_durum, ödül, bitiş = ORTAM.ADIM_AT(aksiyon) # Ortamda aksiyonu uygula
    toplam_ödül = toplam_ödül + ödül # Toplam ödülü güncelle
    GEÇİCİ HAFIZAYA KAYDET (mevcut_durum, aksiyon, ödülü, sonraki_durum, bitiş) # Deneyimleri geçici hafızaya kaydet
```

```
    AJANI EĞİT (geçici_hafiza) # SAC ajanını geçici hafızadaki deneyimlerle eğit
    mevcut_durum = sonraki_durum # Mevcut durumu güncelle
```

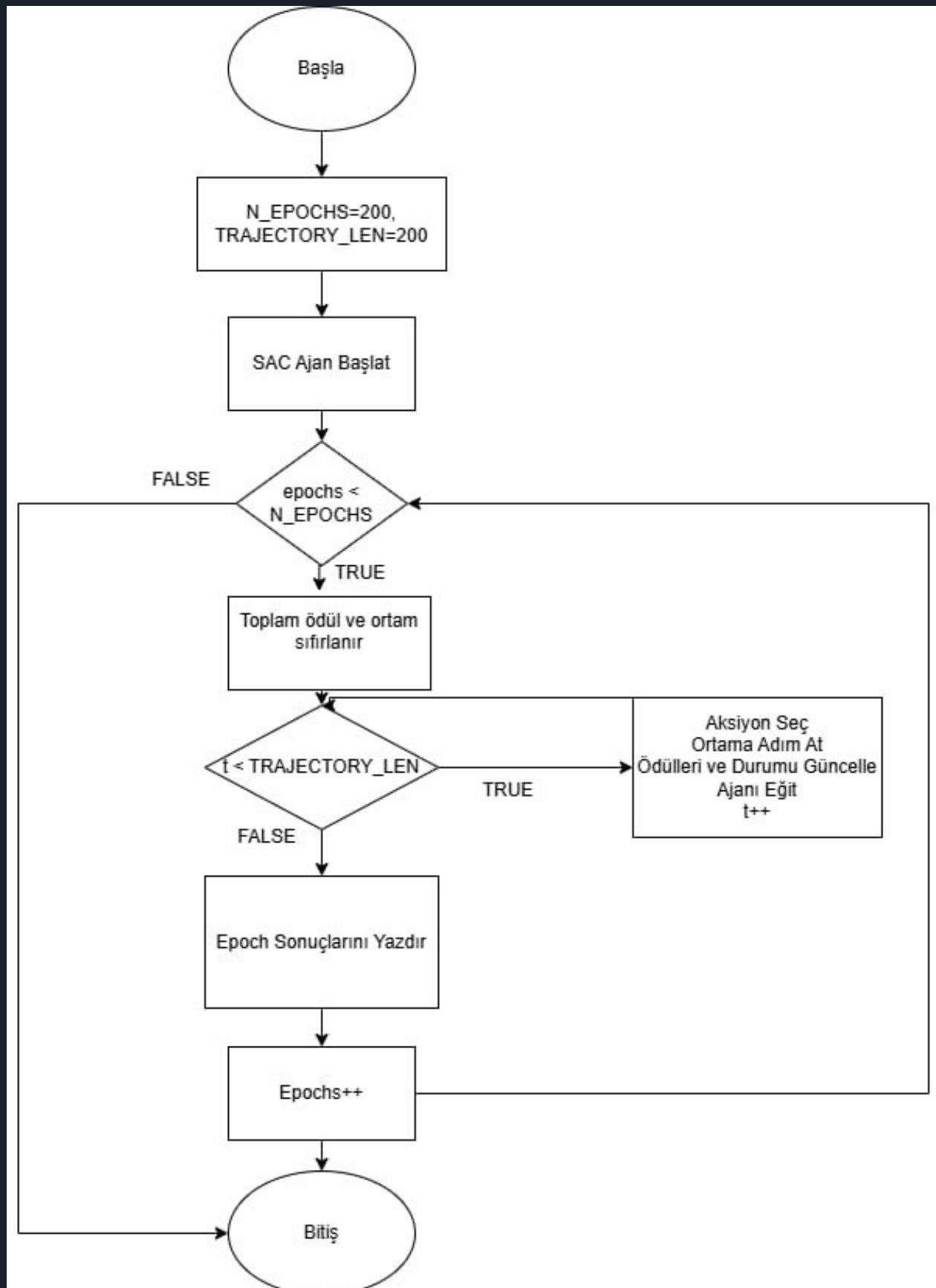
EĞER (bitiş == DOĞRU) İSE:

DÖNGÜDEN ÇIK

```
    t = t + 1
```

```
EPOCH SONUCLARINI YAZDIR (epochs, toplam_ödül) # Her epoch sonunda toplam ödülü yazdır
epochs = epochs + 1
```

BITİR



Deep Deterministic Policy Gradient (DDPG)

DDPG, sürekli eylem alanları için geliştirilmiş bir politika gradyan yöntemidir. Çevreye belirli bir eylemi sürekli olarak uygulayan bir "actor" ve bu eylemleri değerlendiren bir "critic" ile çalışır. DDPG, Q-learning ve politika gradyanı yöntemlerini birleştirerek, yüksek boyutlu eylem alanları için optimize edilmiştir. Özellikle robot manipülasyon görevlerinde başarı göstermektedir.

Deep Deterministic Policy Gradient (DDPG)

Q-değeri, DDPG'de kritik bir rol oynar ve politika bu değeri maksimize edecek şekilde güncellenir.

DDPG'de, Q-değeri, Bellman denklemine göre hesaplanır:

Bu denklemin parçaları:

- $r(s,a)$: Aksiyon sonrası elde edilen ödül.
- γ : Gelecekteki ödüllerin bugünkü değere indirgeyen çarpan.
- $Q(s',a')$: Bir sonraki durumda, yeni aksiyona göre tahmin edilen Q-değeri.

DDPG, hedef bir Q ağı kullanarak, Q-değeri tahminlerinde stabilité sağlar. Q-değeri güncellemesinde, hedef ağıın çıktıları kullanılarak öğrenme daha kararlı hale getirilir.

DDPG Sözde Kodu ve UML Diyagramı

```
N_EPOCHS = 200
TRAJECTORY_LEN = 200
ORTAMI BAŞLAT
DDPG AJANI BAŞLAT
epochs = 0

DÖNGÜ (epochs < N_EPOCHS) DO:
    toplam_ödül = 0
    mevcut_durum = ORTAMI SIFIRLA
    t = 0

    DÖNGÜ (t < TRAJECTORY_LEN) DO:
        AKSIYON SEÇ (mevcut_durum, ddpg_agent + gürültü) # DDPG ajanı mevcut duruma göre aksiyon seçer ve gürültü eklenir
        sonraki_durum, ödül, bitiş = ORTAM.ADIM_AT(aksiyon) # Ortamda aksiyonu uygula
        toplam_ödül = toplam_ödül + ödül # Toplam ödül güncelle
        GEÇİCİ HAFIZAYA KAYDET (mevcut_durum, aksiyon, ödül, sonraki_durum, bitiş) # Deneyimleri geçici hafızaya kaydet

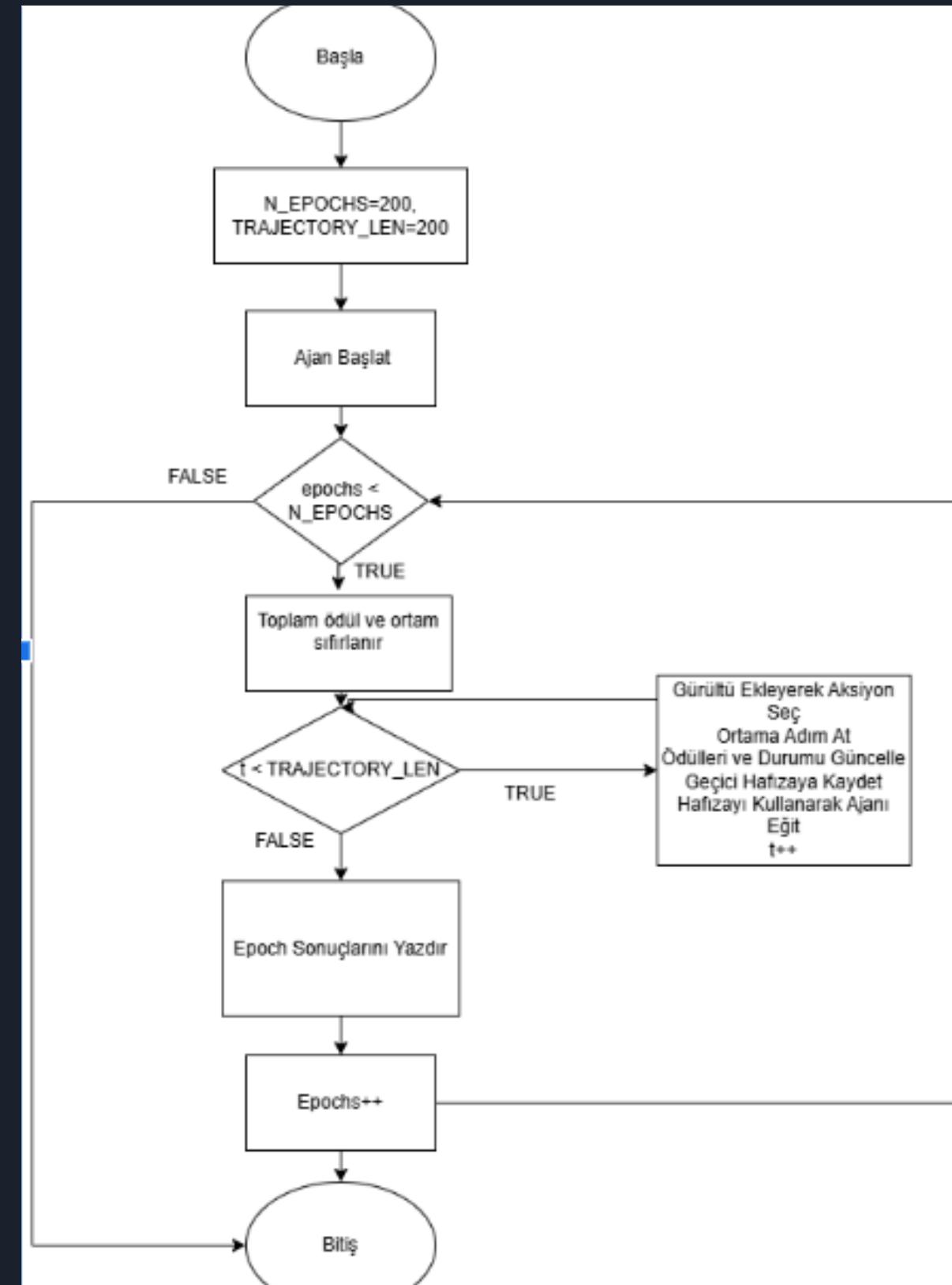
        AJANI EĞİT (geçici_hafıza) # DDPG ajanını geçici hafızadaki deneyimlerle eğit
        mevcut_durum = sonraki_durum # Mevcut durumu güncelle

        EĞER (bitiş == DOĞRU) İSE:
            DÖNGÜDEN ÇIK

        t = t + 1

    EPOCH SONUÇLARINI YAZDIR (epochs, toplam_ödül) # Her epoch sonunda toplam ödül yazdır
    epochs = epochs + 1

BITİR
```



Q-Değeri Hesaplama Farkları

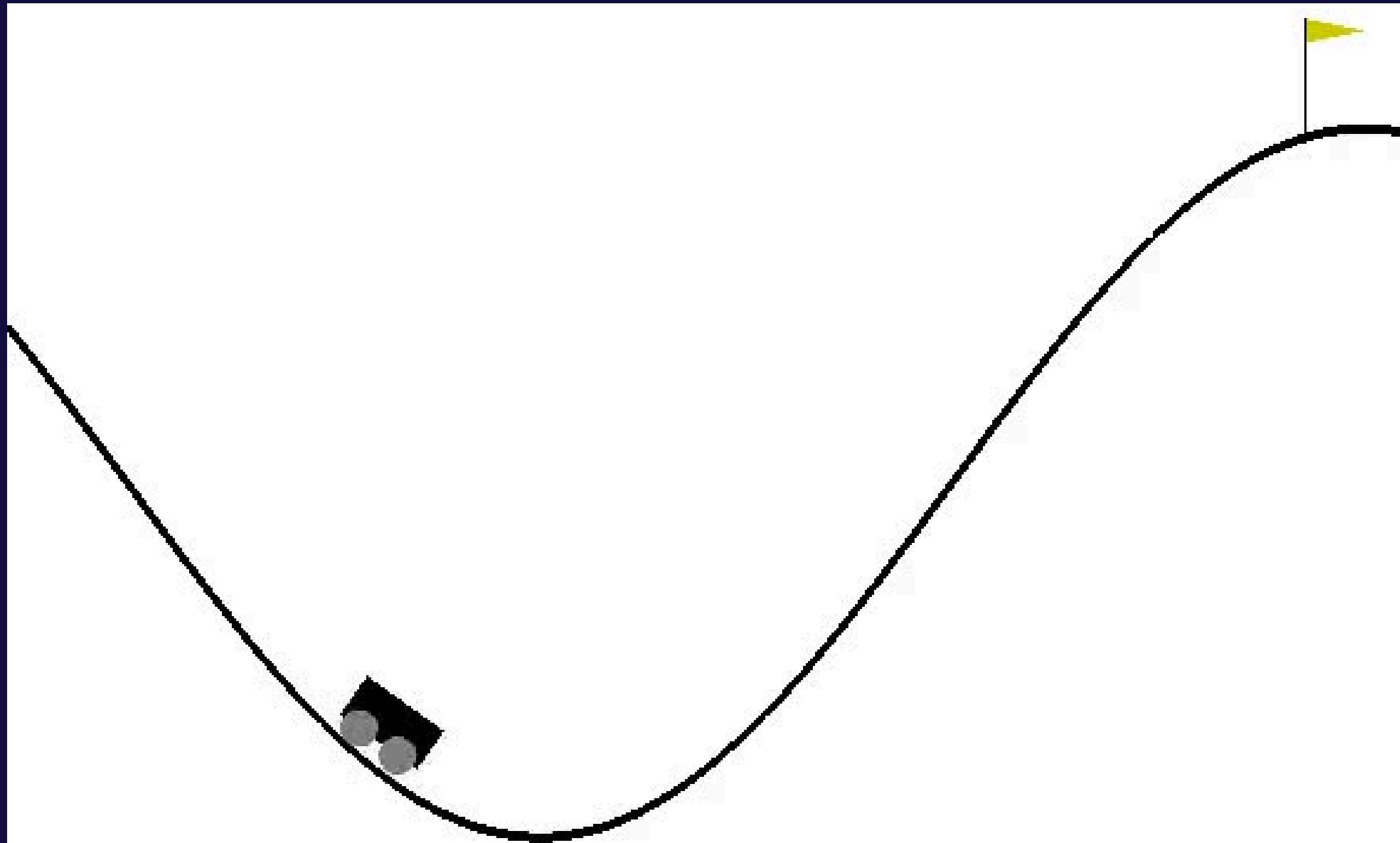
Algoritma	Q-Değeri Hesaplama	Farklılıklar
SAC	$Q(s, a) = r(s, a) + \gamma \cdot V(s')$	İki Q ağı kullanır, entropiyi maksimize ederek keşfi teşvik eder
PPO	$A(s, a) = Q(s, a) - V(s)$ (avantaj fonksiyonu ile)	Q-değerini doğrudan kullanmaz, avantaj fonksiyonuna odaklanır
DDPG	$Q(s, a) = r(s, a) + \gamma \cdot Q(s', a')$	Deterministik aksiyon seçimi yapar ve hedef Q ağı kullanarak stabilité sağlar

Mountain Car Problemi

Mountain Car, iki tekerlekli bir arabayı bir vadiden yukarı taşımak üzerine kurulu bir kontrol problemidir. Amaç, arabanın yeterli hızla ulaşarak vadinin sağ yamacına tırmanmasını sağlamaktır. Ancak, araç motorunun yeterli gücü olmadığı için doğrudan tepeye çıkamaz; vadinin iki yamacını kullanarak momentum oluşturmaya gereklidir. Bu problem, yetersiz güç kaynaklarına sahip robotik sistemlerin verimli bir şekilde görev gerçekleştirebilmesini test eden senaryolara benzemektedir.



Mountain Car Problemi



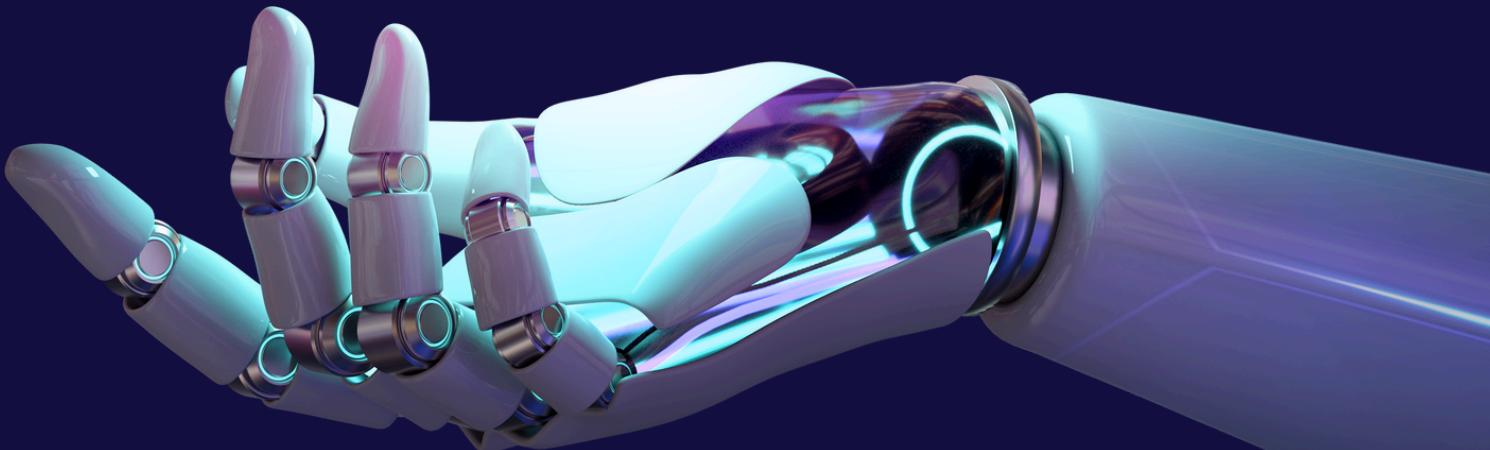
Seçilen Algoritma ve Problem Hakkında Yapılan Çalışmalar

PPO: Mountain Car probleminde PPO'nun sınırlayıcı politika güncellemeleri, özellikle vadinin iki tarafındaki hareketlerdeki küçük değişiklikler nedeniyle avantaj sağlar.

SAC: Mountain Car probleminde yeni hareketler keşfetmesini kolaylaştırır. Özellikle entropi temelli ödüllendirme, aracın vadinin sol tarafına doğru tekrar gitmesini teşvik ederek maksimum momentum kazanmasını sağlar.

DDPG: Mountain Car probleminde aracın momentum kazanmak için ihtiyaç duyduğu hassas hareketleri belirleyebilir. DDPG'nin aktör-eleştirmen yapısı, aracın her iki yamaçtaki hareketini değerlendirerek hassas manevraları öğrenmesini sağlar.

Bu algoritmaların Mountain Car problemine uygulanması, robotik kontrol ve dengeleme problemleri üzerinde başarıyla uygulanabileceğine dair kanıt sağlamaktadır.





Mountain Car mantığı nedir?

Mountain Car problemi, bir vadinin ortasında hareketsiz halde duran bir arabanın, sadece gaz ve fren kontrolü kullanarak vadinin tepesine çıkışmasını sağlamayı amaçlar. Arabanın motor gücü sınırlıdır ve bu yüzden doğrudan tepeye tırmanamaz. Ajan, vadinin bir tarafında hız kazanarak diğer tarafa geçecek şekilde hareket eder ve bu salınım hareketi ile tepeye ulaşmaya çalışır.

Mountain Car sınırları nedir?

Gözlem Alanı:

- Arabanın pozisyonu hızı ile belirlenir. Pozisyon, vadinin iki tarafı arasında sınırlıdır.
- Pozisyon: $[-1.2, 0.6]$ arasında bir değerdir.
- Hız: $[-0.07, 0.07]$ arasında değişir.

Eylem Alanı:

- Araba için üç eylem seçeneği vardır: sola git (gaz), sağa git (gaz) ve durmak. Bu, Mountain Car problemini ayrık (discrete) hale getirir.

Ödül Yapısı:

- Tepeye ulaşmak ve belirlenen hedefi geçmek için ödüller verilmekte, aksi halde her bir adımda cezalandırılmaktadır.
- Genellikle ajan, tepeye ulaştığında pozitif bir ödül, diğer durumlarda her adım için küçük bir negatif ödül alır. Bu, ajanı hedefe daha hızlı ulaşması için teşvik eder.

Sonlandırma Kriteri:

- Ajan belirlenen pozisyon'a (tepeye) ulaştığında veya maksimum adım sınırlarına geldiğinde görev sona erer.



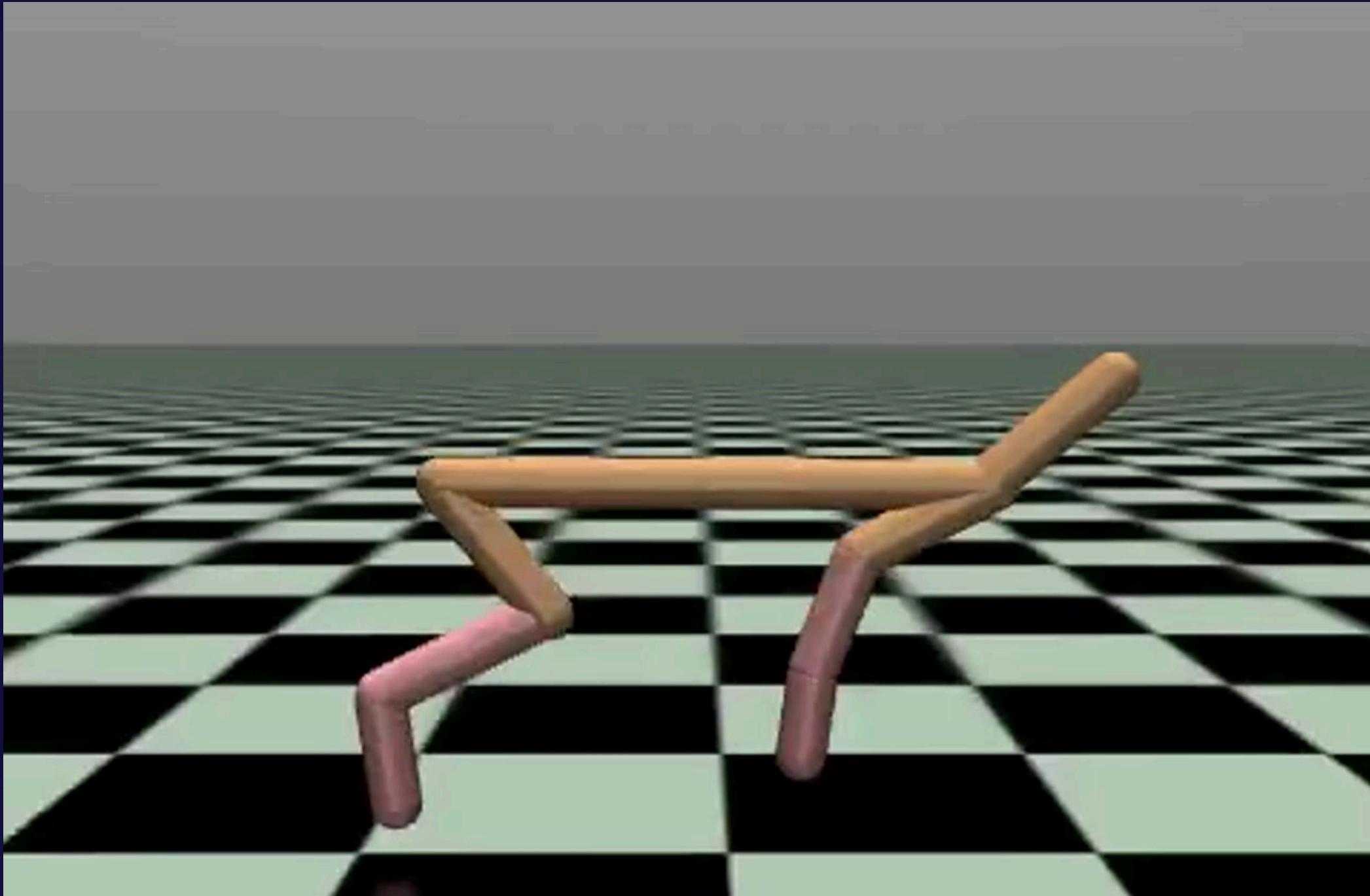
Mountain Car ayrık mı , sürekli mı?

Mountain Car probleminin eylem alanı ayrık bir yapıya sahiptir. Ajanın üç farklı eylemden birini seçebilmesi, problemi ayrık hale getirir. Ancak, bazı uygulamalarda bu problem sürekli eylem alanına genişletilebilir, bu durumda araba hızını sürekli bir şekilde ayarlayabilir. Ancak klasik Mountain Car ortamında, eylemler ayrık olarak tanımlanır.

Half Cheetah Problemi

Half Cheetah problemi, robotik bir ajan olan "yarım çita" modelinin ileriye doğru koşması gereken klasik bir sürekli kontrol problemidir. Ajan, omurgası boyunca eklemelere sahip bir yarı çita gibi tasarlanmıştır ve amacı, belirli bir yöne en hızlı ve en dengeli şekilde ilerlemek için optimal bir hareket stratejisi geliştirmektir. Bu görevde, ajan hızlanmaya çalışırken hem dengeyi koruması hem de enerjiyi verimli kullanması gereklidir. Half Cheetah, sürekli kontrol problemi olarak adlandırılan problemlerin bir örneğidir ve RL algoritmalarının karmaşık hareket kontrolünü öğrenip öğrenemediğini test etmek için yaygın olarak kullanılır.

Half Cheetah Problemi



Seçilen Algoritma ve Problem Hakkında Yapılan Çalışmaları

HPPO (Proximal Policy Optimization): Half Cheetah gibi sürekli kontrol gerektiren ortamlarda PPO, politika tabanlı bir algoritma olarak güvenli ve dengeli öğrenmeyi sağlar. PPO'nun temel avantajı, kontrol edilen güncellemelerle modelin kararlılığını artırmasıdır; bu sayede ani davranış değişikliklerinden kaçınır. Half Cheetah ortamında, PPO'nun sağladığı bu kararlılık sayesinde, ajan her adımda dengeyi koruyarak hız kazanabilir ve sürekli iyileşen bir hareket stratejisi geliştirebilir. PPO'nun bu özellikleri, eğitim sürecini daha güvenilir hale getirir.

Seçilen Algoritma ve Problem Hakkında Yapılan Çalışmaları

DDPG (Deep Deterministic Policy Gradient): DDPG, sürekli aksiyon uzayında hareket eden ajanlar için tasarlanmış, aktör-kritik yapısına sahip bir algoritmadır. Half Cheetah gibi karmaşık ortamlarda, DDPG'nin deterministik politika yapısı, ajan için en iyi hareketi belirlemeye odaklanır ve uzun vadede istikrarlı hareketler öğrenmesine yardımcı olur. DDPG'nin bu avantajı, ajanı hızla ileri taşırken dinamik dengenin korunmasına olanak tanır. Bu algoritmanın Half Cheetah ortamında enerji verimliliğine katkı sağlayarak yüksek performans elde etmesine yardımcı olur.

Seçilen Algoritma ve Problem Hakkında Yapılan Çalışmaları

SAC (Soft Actor-Critic): SAC, entropi bazlı bir algoritma olarak, Half Cheetah ortamında ajan için daha keşif odaklı bir strateji sağlar. Entropiyi artırarak ajanı yeni ve farklı hareketleri keşfetmeye teşvik eder. Bu durum, ajan için hareket stratejilerini zenginleştirir ve çevresel değişikliklere daha esnek yanıtlar verebilmesine yardımcı olur. SAC, Half Cheetah ortamında ajanın maksimum performansa ulaşırken denge ve hız kombinasyonunu optimize etmesine katkıda bulunur ve adaptif öğrenme süreci sağlar.

Half Cheetah mantığı nedir?

Half Cheetah ortamının temel mantığı, bir robotik ajanın iki boyutlu bir yüzeyde en hızlı ve en dengeli şekilde ileriye hareket etmesini sağlamaktır. Ajan, çita gibi koşmaya çalışan, gövdesi ve bacakları eklemli bir yapıya sahip bir modeldir. Bu modelin amacı, belirli eklemlere kuvvet uygulayarak dengeli bir koşu hareketi oluşturmak ve ileri doğru maksimum hızla ilerlemektir. Ödüller, ajanın ileriye hareket etme hızına ve genel performansına göre verilir; bu da ajanın enerji verimliliği ve dengesi gibi faktörleri optimize ederek en uygun hareket stratejisini geliştirmesine yönlendirir. Bu ortam, sürekli kontrol problemleri için RL algoritmalarının performansını ve adaptasyon yeteneklerini test etmek için idealdir.

Refer to learning algorithms in the best-performing algorithm

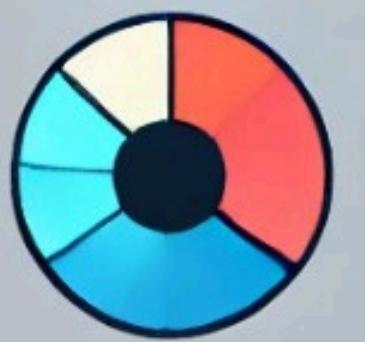
HALF CHEETAH



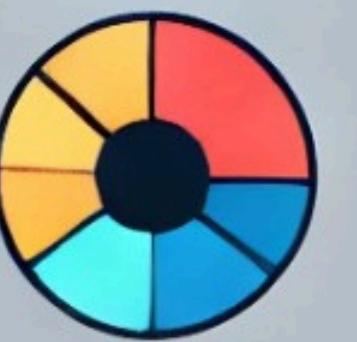
DDPG



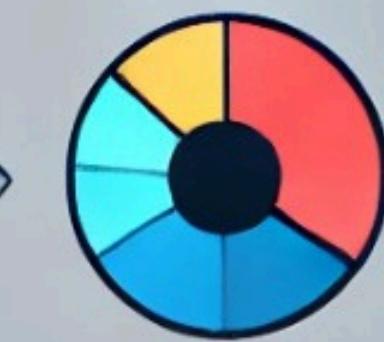
PPG



PPO



SAC



HALF CHEETAH



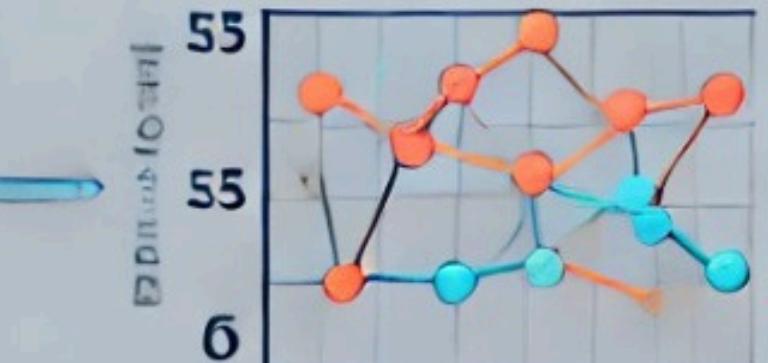
SSAUIE



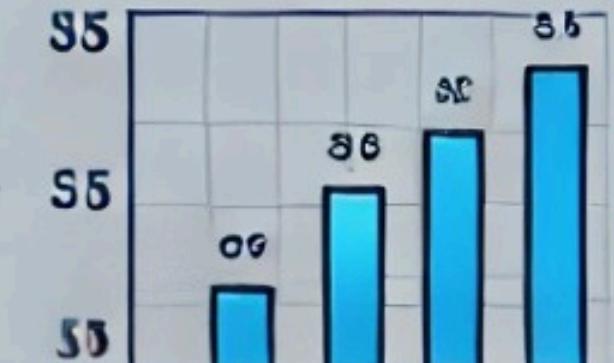
DDPG



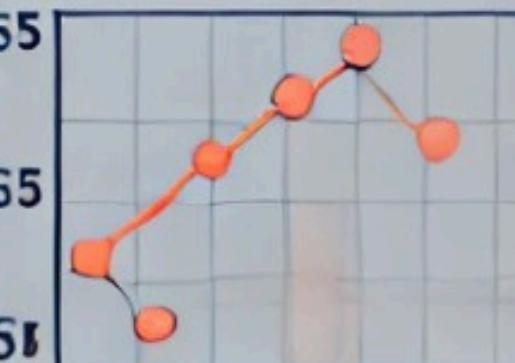
PPO



SAC

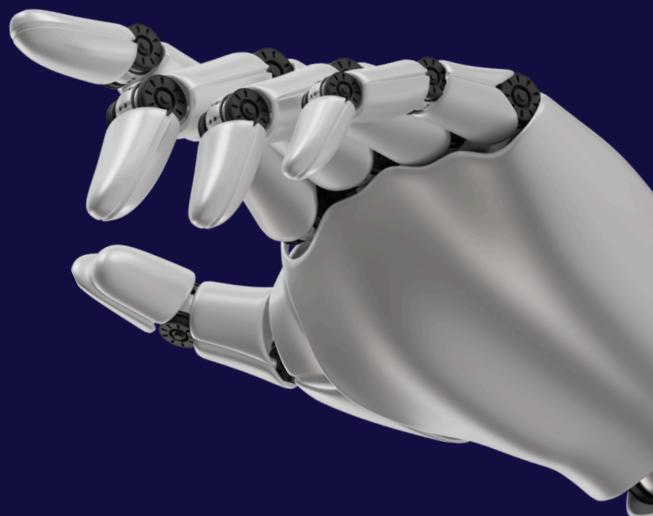


SSAUIE



Half Cheetah ayrık mı, sürekli mı?

Half Cheetah ortamı sürekli bir aksiyon uzayına sahiptir. Bu, ajanın hareket ederken eklemlerine belirli bir kuvvet uygulayarak her adımdaki hareketini sürekli olarak kontrol edebilmesi anlamına gelir. Ajan, farklı eklemelere belirli bir kuvvet aralığında komut gönderir ve bu kuvvet değerleri sürekli bir spektrumda olabilir. Bu tür sürekli aksiyon uzayına sahip ortamlar, daha ince ayarlanmış ve kesintisiz hareket gerektiren görevler için uygundur ve robotik hareket kontrolü gibi uygulamalarda oldukça önemlidir.



KAYNAKÇA

- Sutton, R. S., & Barto, A. G. (2018). Reinforcement Learning: An Introduction (Second Edition). MIT Press.
- Reinforcement learning algoritmalarının temelleri, PPO, SAC ve DDPG gibi algoritmaların genel yapısı.
- Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal Policy Optimization Algorithms. OpenAI. PPO algoritmasının teorik açıklamaları ve çalışma prensipleri.
- Haarnoja, T., Zhou, A., Abbeel, P., & Levine, S. (2018). Soft Actor-Critic Algorithms and Applications. SAC algoritmasının detayları ve uygulamaları.
- Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., ... & Wierstra, D. (2015). Continuous control with deep reinforcement learning. DDPG algoritmasının teorik temelleri ve performans analizi.
- OpenAI Gym Documentation. Mountain Car Environment. OpenAI Gym, Mountain Car probleminin tanımı ve ortam ayarları.
- Stable-Baselines3 Documentation. PPO, SAC ve DDPG algoritmalarının Python uygulamaları ve kod örnekleri.

KAYNAKÇA

- Sutton, R. S., & Barto, A. G. (2018). Reinforcement Learning: An Introduction (Second Edition). MIT Press.
- Reinforcement learning algoritmalarının temelleri, PPO, SAC ve DDPG gibi algoritmaların genel yapısı.
- Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal Policy Optimization Algorithms. OpenAI. PPO algoritmasının teorik açıklamaları ve çalışma prensipleri.
- Haarnoja, T., Zhou, A., Abbeel, P., & Levine, S. (2018). Soft Actor-Critic Algorithms and Applications. SAC algoritmasının detayları ve uygulamaları.
- Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., ... & Wierstra, D. (2015). Continuous control with deep reinforcement learning. DDPG algoritmasının teorik temelleri ve performans analizi.
- OpenAI Gym Documentation. Mountain Car Environment. OpenAI Gym, Mountain Car probleminin tanımı ve ortam ayarları.
- Stable-Baselines3 Documentation. PPO, SAC ve DDPG algoritmalarının Python uygulamaları ve kod örnekleri.

Dinlediğiniz
için
Teşekkürler !

