## İLKİNİN SÖZDE KODU

## PPO Eğitimi İçin Sözde Kod

Başlangıç:

- Gerekli kütüphaneleri yükle
- PPO ağ yapısını ve ajanı tanımla
- Ortam (gym ortamı gibi) başlat
- Logger sınıfı ile eğitim süreci için log dosyası hazırla

PPO_Agent Sınıfı:

- Ağ yapılandırması ve ileri hesaplama (policy ve value) fonksiyonlarını oluştur
- select_action(policy) fonksiyonu ile olasılık dağılımına göre bir eylem seç

Env_Runner Sınıfı:

- Ortamı, ajanı ve logger'ı alarak başlat
- `run(steps)` fonksiyonu ile:
  - Belirli bir adım sayısınca ortamı çalıştır
  - Ajanın eylem politikasına göre eylemleri seç
  - Her adım için gözlem, ödül ve eylem olasılıklarını kaydet
  - Çevre tamamlanırsa ortamı sıfırla ve toplam getiriyi log'la

Avantaj ve Değer Hedefi Hesapla (compute_advantage_and_value_targets):

- Geriye dönük bir döngü ile:
  - Ödül, değer ve yapılan eylemlerle avantaj değerlerini hesapla
  - Yeniden başlatılan (done) durumları göz önünde bulundurarak avantajları ve değer hedeflerini sırayla ekle

Batch_DataSet Sınıfı:

- Eğitim verilerini depola ve minibatch veri kümesi oluşturmak için kullan

Ana Eğitim Döngüsü:

Başlat:

- Eğitim adımları, öğrenme oranı ve hiperparametreleri tanımla
- PPO ajanını ve optimizasyonu başlat
- Çevre koşucuları (env_runners) oluştur

Eğitim için döngü (iterations):

- Her bir çevre koşucusu için:
  - Çevreyi belirli bir adım kadar çalıştır (Env_Runner.run)
  - Avantaj ve değer hedeflerini hesapla
  - Toplanan veriyi batch olarak topla
- Batch_DataSet ile toplanan verileri minibatch boyutunda yükle

Güncelleme Döngüsü:

- Belirli epoch sayısınca her minibatch için:
  - Gözlemler ve eylemleri al
  - Avantajları normalize et
  - Politika eylem olasılıkları ile olasılık oranı hesapla
  - PPO kayıp fonksiyonlarını hesapla:
    - Klipsli politika kaybı (L_CLIP)
    - Değer kaybı (L_VF)
  - Toplam kaybı geri yürüterek optimize et

PPO ADIM ADIM:

1. Ajanı ve Çevreyi Başlat: Çevreyi başlat, PPO ağını tanımla, ve eğitim sürecini kaydedecek logger'ı oluştur.
2. Çevreyi Koş ve Veri Topla: Belirli bir adım sayısınca çevreyi koş, gözlem, eylem, ödül ve değer tahminlerini topla.
3. Avantaj ve Değer Hedeflerini Hesapla: Geriye dönük bir döngüyle avantaj ve değer hedeflerini hesapla.
4. Veriyi Minibatch Olarak Eğit: Toplanan veriyi minibatch olarak eğitim veri kümesine yükle.
5. Ağ Güncellemesi: PPO kayıp fonksiyonlarına göre, politika ve değer kayıplarını minimize etmek için optimize et.
6. Eğitimi Tekrarla: Belirtilen iterasyonlar boyunca adımları tekrar ederek ajanı eğit.

## İKİNCİNİN SÖZDE KODU

1. TensorFlow ve TensorFlow Probability kütüphanelerini ve diğer gerekli modülleri içe aktar.

2. Eager Execution'ı devre dışı bırak.

3. PPO sınıfını tanımla:

- Sınıfın başlatıcısını (__init__) tanımla:

- Öğrenme oranları (A_LR ve C_LR), durum ve aksiyon boyutları, güncelleme adım sayısı gibi PPO parametrelerini ayarla.

- Aktör ve Kritiği tanımla:

- 'Critic' için:

- İki katmanlı bir sinir ağı oluştur, duruma göre avantaj fonksiyonunu ve kayıp fonksiyonunu hesapla.

- Kritik için optimize edici tanımla.

- 'Actor' için:

- 'pi' (aktör) ve 'oldpi' (önceki aktör) ağlarını oluştur.

- Olasılık dağılımı, aksiyon örnekleme ve güncelleme işlemlerini ayarla.

- Kayıp fonksiyonunu ve optimize ediciyi tanımla.

4. PPO sınıfı içindeki update fonksiyonu:

- Aktör ve kritik ağlarını güncellemek için güncelleme döngülerini tanımla.

- Kritik ağını avantaj değerlerine göre güncelle.

5. Aksiyon ağını oluşturan _build_anet fonksiyonu:

- İki katmanlı sinir ağı kullanarak aksiyon dağılımının ortalama ve standart sapma değerlerini hesapla.

- Gaussian dağılımını kullanarak aksiyon dağılımını döndür.

6. choose_action fonksiyonu:

- Mevcut duruma göre bir aksiyon seç ve geri döndür.

7. get_v fonksiyonu:

- Mevcut duruma göre durum-değerini döndür.

8. Eğitim sürecini tanımla (train_model fonksiyonu):

- Ortamı ('MountainCarContinuous-v0') hazırla ve hiperparametreleri ayarla.
- PPO sınıfını başlat ve eğitim döngüsünü başlat:
  - Her döngü için:
    - Durumu sıfırla, tamponları ve ödülleri sıfırla.
    - Tamamlanana kadar:
      - Ortamı çizdir.
      - Bir aksiyon seç ve keşif için gürültü ekle.
      - Adım at, durumu ve ödülü tamponlara ekle.
      - Her adım için kritik değeri hesapla, ödülleri güncelle ve PPO ağını güncelle.
    - Bölüm ödülünü yazdır.
  - Her 10 bölümde bir modeli kaydet.

**acrobotVREP için DDPG Ajanının Eğitim Sözde Kodu**

1. Kütüphaneleri İçeri Aktar ve Ortamı Ayarla:
    - Gerekli kütüphaneleri içe aktar: Derin öğrenme (model oluşturma için), takviye öğrenmesi ve özel acrobotVREP ortamı için kütüphaneler.
    - Ortam adını tanımla (örneğin, acrobotVREP-v0).
    - Ortamı başlat:
        - Rastgelelik için belirli tohum değerlerini ayarla (örneğin, 1234).
        - Maksimum adım sayısını belirt.
        - Ortamdaki kullanılabilir eylem sayısını elde et.
2. Aktör Modelini Tanımla:
    - Gözlemlere göre eylem üreten Aktör modelini oluştur.
    - Aktör Ağ Mimarisi:
        - Giriş (gözlem alanını) düzleştir (Flatten).
        - ReLU aktivasyonlu tam bağlı (dense) katmanlar ekle.
        - Çıkış katmanı, eylem sayısıyla eşleşmeli ve doğrusal aktivasyonla (linear) sonuçlanmalıdır.
3. Kritik Modelini Tanımla:
    - Eylemin kalitesini değerlendiren Kritik modelini oluştur.
    - Kritik Ağ Mimarisi:
        - İki giriş alır: eylem ve gözlem (DDPG'de Kritik için her ikisi de gereklidir).
        - Gözlem girişini düzleştir.
        - Eylem ve gözlem girdilerini birleştir (concatenate).
        - Birleşik girdileri ReLU aktivasyonlu tam bağlı katmanlardan geçir.
        - Son katman, eylemin kalitesini (Q değeri) temsil eden tek bir çıktı değeri üretir.

4. Ajanı Yapılandır:
    - Deney tekrar oynatma için bir hafıza tamponu oluştur.
    - Keşif amaçlı bir rastgele gürültü süreci tanımla (Ornstein-Uhlenbeck süreci).
    - DDPG ajanını aşağıdaki bileşenlerle kur:
        - Aktör modeli.
        - Kritik modeli.
        - Kritik için eylem girişi.
        - Hafıza tamponu.
        - Keşif için rastgele süreç.
        - Hiperparametreler, örneğin gamma ve öğrenme hızları gibi.
5. Ajanı Derle ve Eğit:
    - DDPG ajanını bir optimizasyon algoritması (örneğin, Adam) kullanarak derle.
    - Ortamda eğitim sürecini başlat:
        - Eğitim adımlarının sayısını ayarla.
        - Görselleştirme ve çıktı seçeneklerini belirle.
        - Bölüm başına maksimum adım sayısını belirle.
6. Model Ağırlıklarını Kaydet (Eğitim Sonrası):
    - Eğitim tamamlandıktan sonra, eğitimli model ağırlıklarını bir dosyaya kaydet.

**ADIM ADIM**

**acrobotVREP için DDPG Ajanının Adım Adım Açıklaması**

1. Kütüphaneleri İçeri Aktar ve Ortamı Ayarla:
    - Kütüphane İçe Aktarma:
        - Projede ihtiyaç duyulan kütüphaneleri (acrobotVREP, numpy, gym, keras, rl) içeri aktarıyoruz.
        - keras model ve katman yapıları, rl (Reinforcement Learning) takviye öğrenmesi ajanları, gym ise ortam tanımlaması için kullanılır.
    - Ortam Tanımlama:
        - ENV_NAME olarak, kullanmak istediğimiz ortamın adını (acrobotVREP-v0) tanımlıyoruz. Bu ortam gym üzerinden çalıştırılacak.
    - Ortamı Başlatma:
        - Ortamı gym.make(ENV_NAME) ile başlatıyoruz.
        - Maksimum bölüm adım sayısını env._max_episode_steps = 200 ile 200 olarak sınırlıyoruz. Bu, her bölümde en fazla 200 adım alınabileceğini belirler.
    - Rastgelelik Ayarı:
        - numpy ve ortamın rastgelelik ayarlarını seed(1234) ile sabitliyoruz, böylece her çalıştırmada aynı sonuçları alabiliriz.
    - Aksiyon Sayısını Belirleme:
        - env.action_space.shape[0] kullanarak ortamın eylem alanındaki eylem sayısını (kaç farklı aksiyon alabileceğini) nb_actions değişkenine atıyoruz. Bu sayede aktör modeli çıktı katmanı doğru ayarlanacak.
2. Aktör Modelini Tanımla:
    - Aktör Modelinin İşlevi:
        - Aktör, gözlemlerden aksiyonlar çıkartmak için kullanılan bir sinir ağıdır. Bu model, ortamdan aldığı gözlemi işleyerek bir aksiyon önerir.

- Ağ Yapısı:
  - Giriş (Flatten):
    - Flatten(input_shape=(1,) + env.observation_space.shape) ile gözlem verisini düzleştiriyoruz. Bu, gözlem verisinin doğru boyutta işlenebilmesini sağlar.
  - Katmanlar:
    - Üç adet tam bağlı (dense) katman ekliyoruz, her birinde 16 nöron ve relu aktivasyon fonksiyonu var. Bu katmanlar, gözlem verisini işleyerek anlamlı özellikler çıkartır.
  - Çıkış Katmanı:
    - Son tam bağlı katman nb_actions sayısında nöron içerir ve linear aktivasyon fonksiyonuna sahiptir. Bu, aktör modelinin eylem önerilerini doğrusal olarak (linear) vermesini sağlar.

3. Kritik Modelini Tanımla:
   - Kritik Modelinin İşlevi:
     - Kritik, aktörün ürettiği aksiyonun kalitesini ölçer. Bu model, eylemin Q-değerini tahmin eder ve hangi aksiyonların iyi olduğunu değerlendirmeye yardımcı olur.
   - Girişler ve Düzleştirme:
     - Kritik model iki girdi alır: action_input ve observation_input. observation_input (gözlem) düzleştirilir.
   - Girdi Birleştirme (Concatenation):
     - Düzleştirilmiş gözlem ve aksiyon birleştirilir. Bu birleşim kritik modelin hem gözlemi hem de aksiyonu dikkate alarak karar vermesini sağlar.
   - Katmanlar:
     - Üç adet 32 nöronlu tam bağlı katman eklenir. Her bir katmanda relu aktivasyonu kullanılır. Bu katmanlar, aksiyon ve gözlem verisinin işlenmesini sağlar.
   - Çıkış Katmanı:

        - Son katman tek bir nörondan oluşur ve linear aktivasyon kullanır. Bu katman, aksiyonun Q-değerini verir.

4. Ajanı Yapılandır:
    - Bellek (Memory) Tanımlama:
        - SequentialMemory(limit=100000, window_length=1) ile ajanın geçmiş deneyimlerini saklayacağı bir bellek oluşturulur. limit, belleğin kapasitesini belirtir, window_length ise her adımda kaç gözlem tutulacağını ayarlar.
    - Rastgele Gürültü Süreci:
        - Ornstein-Uhlenbeck süreci, ajanı keşif (exploration) yapması için rastgele gürültü ekler. theta, mu ve sigma gibi parametrelerle keşif sürecini ayarlıyoruz.
    - DDPG Ajanı Oluşturma:
        - DDPG ajanı, aktör ve kritik modellerini kullanarak yapılandırılır.
        - Önemli Parametreler:
            - gamma: gelecekteki ödüllerin bugünkü ödüllere göre ne kadar önem taşıdığını belirler.
            - target_model_update: hedef model güncellemelerinin hızını ayarlar (örneğin, 1e-3).
            - nb_steps_warmup_critic ve nb_steps_warmup_actor: eleştirici ve aktör için ısınma adım sayısını belirtir (her biri için 100 adım).
5. Ajanı Derle ve Eğit:
    - Derleme (Compile):
        - Adam optimizasyon algoritması ile DDPG ajanı derlenir. metrics=['mae'], hata ölçümünde ortalama mutlak hata (MAE) metriğini kullanır.
    - Eğitim Başlatma:
        - agent.fit ile ajan eğitilir. Eğitim parametreleri:

        - nb_steps=10000: Toplam adım sayısı 10,000 olarak belirlenir.
        - visualize=True: Ortamda görselleştirme etkinleştirilir.
        - verbose=0: Eğitim sırasında çıktı seviyesini belirler.
        - nb_max_episode_steps=200: Her bölümde maksimum 200 adım alınabileceğini belirtir.
6. Model Ağırlıklarını Kaydet (Eğitim Sonrası):
    - Modeli Kaydetme:
        - agent.save_weights('ddpg_{}_weights.h5f'.format(ENV_NAME), overwrite=True) ifadesi, eğitim tamamlandıktan sonra model ağırlıklarını dosyaya kaydeder.
        - Bu ağırlıklar daha sonra yüklenip kullanılarak ajanın eğitilmiş haliyle çalıştırılabilir.

MOUNTAIN CAR CONTINUOUS

```
# Gerekli kütüphaneleri içe aktar
# argparse: Komut satırı argümanlarını okumak için
# gymnasium (gym): Çevreyi oluşturmak için (MountainCarContinuous-v0 kullanıyoruz)
# numpy: Sayısal işlemler için
# matplotlib.pyplot: Grafik çizmek için
# pickle: Q tablosunu kaydetmek ve yüklemek için


# Fonksiyon: run
# Parametreler:
# - is_training (True olduğunda eğitim yapar, False olduğunda modeli test eder)
# - render (True olduğunda çevreyi ekranda gösterir, False olduğunda gizler)


Fonksiyon run(is_training=True, render=False):
    # Çevreyi oluştur
    env = gym.make('MountainCarContinuous-v0', render_mode='human' ise render True aksi halde None)


    # Hiperparametreler
    öğrenme_hızı = 0.9999      # Q tablosunu güncellerken kullanılır
    indirim_faktörü = 0.9      # Gelecekteki ödülleri hesaplarken kullanılır
    epsilon = 1                # Başlangıçta %100 rastgele eylem yapılır
    epsilon_azalma_oranı = 0.001   # epsilon'un azalmasını sağlar
    epsilon_min = 0.05         # epsilon'un ulaşabileceği minimum değer
    pos_aralıkları = 20        # Pozisyonu sürekli durumdan ayrık duruma bölmek için
    vel_aralıkları = 20        # Hızı sürekli durumdan ayrık duruma bölmek için
```

act_aralıkları = 10 # Eylem alanını sürekli durumdan ayrık duruma bölmek için

# Pozisyon ve hız aralıklarını ayarla

pos_space = Pozisyon aralıklarını -1.2 ile 0.6 arasında 20 parçaya böl

vel_space = Hız aralıklarını -0.07 ile 0.07 arasında 20 parçaya böl

# Eylem alanını ayrık segmentlere böl

act_space = Eylem aralıklarını -1.0 ile 1.0 arasında 10 parçaya böl

act_lookup_space = act_space'a 1 ekle (Eylemleri bulmak için)

Eğer (is_training ise):

Q tablosunu (21x21x11 boyutunda sıfırlardan oluşan bir matris olarak) başlat

Aksi halde:

'mountain_car_cont.pkl' dosyasından Q tablosunu yükle

En iyi ödül = -999999 # En iyi ödülü izlemek için

En iyi ortalama ödül = -999999 # En iyi ortalama ödülü izlemek için

rewards_per_episode = [] # Her bölüm için ödülleri saklamak için liste

epsilon_history = [] # epsilon'un her bölümdeki değişimlerini saklayan liste

i = 0 # Bölüm sayacı

Sonsuz döngü (while):

# Başlangıç pozisyonu ve hızını al (her bölüm başında)

state = env.reset()

# Pozisyon ve hızı ayrık duruma çevir

state_p = Pozisyonu ayrık pozisyon aralıklarına göre bul

state_v = Hızı ayrık hız aralıklarına göre bul

terminated = False  # Hedefe ulaşıldığında True olacak

rewards = 0

steps = 0

# Bölüm sonlanana veya adım sayısı 5000'i geçene kadar döngü (while):

Eğer (is_training ve rastgele sayı epsilon'dan küçükse):

    # Rastgele bir eylem seç

    action = Rastgele eylem seç

    # Eylem dizinini bul

    action_idx = Eylemi ayrık eylem aralıklarına göre bul

Aksi halde:

    # En yüksek Q değerine sahip eylemi seç

    action_idx = Q tablosunda state_p ve state_v için en yüksek değere sahip eylemi bul

    # Ayrık eylemi sürekli değere dönüştür

    action = act_lookup_space[action_idx]

# Eylemi uygula ve yeni durumu al

yeni_durum, ödül, terminated, _, _ = env.step([action])

# Yeni durumu ayrık duruma çevir

new_state_p = yeni durumu pozisyon aralıklarına göre bul

new_state_v = yeni durumu hız aralıklarına göre bul

```
Eğer (is_training ise):
    # Q tablosunu güncelle
    q[state_p, state_v, action_idx] = Güncelleme formülünü uygula

# Durumu güncelle
state = yeni_durum
state_p = new_state_p
state_v = new_state_v

# Ödülleri toplar
rewards += ödül

steps += 1

# En iyi ödülü güncelle
Eğer ödüller > en iyi ödül ise:
    en iyi ödül = ödüller

# Bölüm başına ödülleri kaydet
rewards_per_episode.append(rewards)

Eğer (is_training ve i != 0 ve i%100 == 0 ise):
    # Son 100 bölümün ortalama ödülünü hesapla
    mean_reward = Son 100 bölümün ortalamasını bul
    Ortalamayı ve epsilon değerini çiz ve kaydet
```

```
            Eğer (mean_reward > en iyi ortalama ödül):
                en iyi ortalama ödül = mean_reward
                Modeli kaydet

            Eğer mean_reward > hedef ödül eşiği ise:
                döngüden çık

        Aksi halde, (is_training değilse):
            Bölüm ödülünü yazdır

        # epsilon'u azalt
        epsilon = max(epsilon - epsilon_azalma_oranı, epsilon_min)
        epsilon_history.append(epsilon)

        # Bölüm sayacını artır
        i += 1

    Çevreyi kapat



# Komut satırı argümanlarını ayarla
Eğer --test bayrağı belirtilmişse:
    run(is_training=False, render=True)
Aksi halde:
    run(is_training=True, render=False)
```

ADIM ADIM

1. Kütüphaneleri İçe Aktarma

```
import argparse
import gymnasium as gym
import numpy as np
import matplotlib.pyplot as plt
import pickle
```

Bu bölümde:

- argparse: Komut satırı argümanlarını okumak için kullanılıyor.
- gymnasium: OpenAI'nin gym ortamını çalıştırmak için kullanılıyor (gym ortamı, çeşitli simülasyonları sağlar).
- numpy: Matematiksel işlemler ve veri yapıları için kullanılıyor.
- matplotlib.pyplot: Eğitim sürecindeki ödülleri ve epsilon değerlerini grafikle göstermek için kullanılıyor.
- pickle: Q-tablosunu kaydetmek ve yüklemek için kullanılıyor.

2. run Fonksiyonu

```
def run(is_training=True, render=False):
```

Bu fonksiyon, ajanı çalıştıran ve eğiten ana fonksiyon. is_training parametresi, eğitim modunda olup olmadığını belirliyor. render parametresi ise simülasyon ekranını görselleştirip görselleştirmemeyi kontrol ediyor.

3. Ortamı Başlatma

```
env = gym.make('MountainCarContinuous-v0', render_mode='human' if render else None)
```

MountainCarContinuous-v0 adlı ortam oluşturuluyor. render True olduğunda, ortam görsel olarak gösteriliyor.

4. Hiperparametreleri Tanımlama

```
learning_rate_a = 0.9999
discount_factor_g = 0.9
```

epsilon = 1

epsilon_decay_rate = 0.001

epsilon_min = 0.05

pos_divisions = 20

vel_divisions = 20

act_divisions = 10

Hiperparametreler:

- learning_rate_a: Q-tablosu güncellemelerinde öğrenme hızı.
- discount_factor_g: Gelecekteki ödüllerin önemini belirler.
- epsilon: Başlangıçta 1, ajan %100 rastgele eylemler seçiyor. Ajan eğitim aldıkça epsilon azalıyor.
- epsilon_decay_rate: Her bölümde epsilon'un ne kadar azalacağını belirler.
- epsilon_min: Epsilon'un ulaşabileceği minimum değer.
- pos_divisions ve vel_divisions: Konum ve hız değerleri için ayrılan bölümlerin sayısı.
- act_divisions: Aksiyon alanının bölümlerinin sayısı.

5. Sürekli Durum ve Aksiyon Uzayını Bölümlere Ayırma

pos_space = np.linspace(env.observation_space.low[0], env.observation_space.high[0], pos_divisions)

vel_space = np.linspace(env.observation_space.low[1], env.observation_space.high[1], vel_divisions)

act_space = np.linspace(env.action_space.low[0], env.action_space.high[0], act_divisions, endpoint=False)

act_lookup_space = np.append(act_space, 1)

Burada sürekli uzay, ayrık (discrete) bir uzaya bölünüyor:

- pos_space ve vel_space: Konum ve hız değerleri için aralıklar oluşturuyor.
- act_space ve act_lookup_space: Aksiyon aralıkları ve eylemleri ayrık hale getiriyor.

6. Q-Tablosunu Başlatma

```
if(is_training):
    q = np.zeros((len(pos_space)+1, len(vel_space)+1, len(act_space)+1))
else:
    f = open('mountain_car_cont.pkl', 'rb')
    q = pickle.load(f)
    f.close()
```

Eğer is_training True ise, sıfırlarla dolu bir Q-tablosu oluşturuluyor. Aksi halde, önceden eğitilmiş bir Q-tablosu dosyadan yükleniyor.

7. Eğitim Döngüsü

```
while(True):
```

Ana döngü başlatılıyor. Bu döngü, her bölümü ayrı ayrı çalıştırarak ajanı eğitiyor veya test ediyor.

8. Başlangıç Durumunu ve Ayrıklaştırılmış Değerleri Alma

```
state = env.reset()[0]
state_p = np.digitize(state[0], pos_space)
state_v = np.digitize(state[1], vel_space)
```

Her bölümün başında, başlangıç durumu sıfırlanıyor ve konum ve hız değerleri ayrık değerlere dönüştürülüyor.

9. Bölüm Sonlanana Kadar Adımları Yürütme

```
while(not terminated and steps<5000):
```

Her bölüm için ajan, en fazla 5000 adım boyunca hareket eder. Bu sırada ajan ya rastgele bir aksiyon seçer ya da Q-tablosuna göre en iyi aksiyonu seçer.

a. Rastgele veya Q-Tablosuna Göre Aksiyon Seçme

```
if is_training and np.random.rand() < epsilon:
    action = env.action_space.sample()
    action_idx = np.digitize(action, act_space)
else:
```

action_idx = np.argmax(q[state_p, state_v, :])

action = act_lookup_space[action_idx]

- is_training True ve epsilon değeri rastgele sayıdan büyükse, rastgele bir aksiyon seçiliyor.
- Aksi durumda, Q-tablosuna göre en yüksek değere sahip aksiyon seçiliyor.

b. Aksiyon Uygulama ve Yeni Durumu Güncelleme

new_state, reward, terminated, _, _ = env.step(np.array([action]))

new_state_p = np.digitize(new_state[0], pos_space)

new_state_v = np.digitize(new_state[1], vel_space)

Seçilen aksiyon uygulanıyor, yeni durum ve ödül alınıyor. Yeni durum da ayrık değerlere dönüştürülüyor.

c. Q-Tablosunu Güncelleme

if is_training:

q[state_p, state_v, action_idx] = q[state_p, state_v, action_idx] + learning_rate_a * (

reward + discount_factor_g * np.max(q[new_state_p, new_state_v, :]) - q[state_p, state_v, action_idx]

)

Eğer eğitim modundaysa, Q-değeri güncelleniyor. Bu işlem, Q-öğrenme algoritmasını kullanarak yapılır.

d. Durumları Güncelleme ve Ödülleri Toplama

state = new_state

state_p = new_state_p

state_v = new_state_v

rewards += reward

steps += 1

Ajan yeni duruma geçiyor ve ödüller toplanıyor.

10. Epsilon Azaltma ve Bölüm İstatistiklerini Kaydetme

```
epsilon = max(epsilon - epsilon_decay_rate, epsilon_min)
epsilon_history.append(epsilon)
```

Epsilon değeri, her bölümde azalıyor ve minimum değere ulaştığında daha fazla azalması engelleniyor.

11. Modeli Kaydetme ve Grafikleme

Her 100 bölümde bir, ortalama ödül hesaplanarak gösteriliyor. Eğer ortalama ödül daha önceki en iyi ortalama ödülden yüksekse, model kaydediliyor ve eğitim süreci durduruluyor.

```
if mean_reward > best_mean_reward:
    f = open('mountain_car_cont.pkl', 'wb')
    pickle.dump(q, f)
    f.close()
```

12. Ana Program

```
if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument('--test', action='store_true', help='Test model')
    args = parser.parse_args()
    if args.test:
        run(is_training=False, render=True)
    else:
        run(is_training=True, render=False)
```

Komut satırında --test argümanı verilirse, model is_training=False olarak test modunda çalıştırılır.