

**심층강화학습 기말과제**

**과목명 심층강화학습**

**담당교수님 최재영교수님**

**제출일 20241210**

**전공 컴퓨터공학과**

**학번 202430026**

**이름 이준용**

* **Implementing Code 2.1 and Code 2.10 in the textbook**

**▪ Analyzing and interpreting codes line by line**

**▪ Draw graph in Figure 2.2 (textbook) and explaining the results**

**Code 2.1 코드 분석**

from torch.distributions import Categorical

import gym

import numpy as np

import torch

import torch.nn as nn

import torch.optim as optim

# 할인율 설정 (미래 보상에 대한 할인 계수)

gamma = 0.99

# 정책 신경망 클래스 정의 (정책 네트워크)

class Pi(nn.Module):

    def \_\_init\_\_(self, in\_dim, out\_dim):

        super(Pi, self).\_\_init\_\_()

        # 신경망 계층 정의 (입력 -> 은닉층(64) -> 출력층)

        layers = [

            nn.Linear(in\_dim, 64), # 입력층에서 은닉층으로의 선형 변환

            nn.ReLU(), # 비선형 활성화 함수 ReLU

            nn.Linear(64, out\_dim), # 은닉층에서 출력층으로의 선형 변환

        ]

        self.model = nn.Sequential(\*layers) # 계층을 하나의 신경망으로 구성

        self.onpolicy\_reset() # 정책 초기화 (에피소드 시작 시)

        self.train() # 학습 모드로 설정

    def onpolicy\_reset(self):

        # 정책에 대한 로그 확률 및 보상 초기화 (에피소드마다 리셋)

        self.log\_probs = []

        self.rewards = []

    def forward(self, x):

        # 상태 x를 입력으로 받아 확률 분포의 파라미터를 출력

        pdparam = self.model(x)

        return pdparam

    def act(self, state):

        # 환경의 상태를 입력으로 받아 행동을 샘플링

        x = torch.from\_numpy(state.astype(np.float32)) # numpy 배열을 torch 텐서로 변환

        pdparam = self.forward(x) # 정책 네트워크의 순전파

        pd = Categorical(logits=pdparam) # 카테고리 확률 분포 생성

        action = pd.sample() # 확률 분포에 따라 행동을 샘플링

        log\_prob = pd.log\_prob(action) # 행동의 로그 확률 계산

        self.log\_probs.append(log\_prob) # 로그 확률 저장 (REINFORCE 알고리즘에 필요)

        return action.item() # 행동 반환 (정수 값)

# REINFORCE 알고리즘의 학습 함수

def train(pi, optimizer):

    T = len(pi.rewards) # 에피소드 길이 (전체 타임스텝 수)

    rets = np.empty(T, dtype=np.float32) # 리턴 값 배열 생성

    future\_ret = 0.0 # 미래 보상 초기화

    # 리턴 값(누적 보상)을 뒤에서부터 계산 (종료 상태부터 역으로)

    for t in reversed(range(T)):

        future\_ret = pi.rewards[t] + gamma \* future\_ret # G\_t = r\_t + γ \* G\_t+1

        rets[t] = future\_ret # 현재 타임스텝의 리턴 저장

    rets = torch.tensor(rets) # 리턴 값을 torch 텐서로 변환

    log\_probs = torch.stack(pi.log\_probs) # 로그 확률을 하나의 텐서로 결합

    loss = - log\_probs \* rets # 손실 함수 (로그 확률과 리턴의 곱, 부호 반전)

    loss = torch.sum(loss) # 손실의 합계 계산

    optimizer.zero\_grad() # 기존의 그래디언트 초기화

    loss.backward() # 손실에 대한 그래디언트 계산 (역전파)

    optimizer.step() # 그래디언트를 사용해 정책 네트워크 파라미터 업데이트

    return loss # 손실 반환 (출력 용도)

# 메인 함수 (학습 절차 실행)

def main():

    env = gym.make('CartPole-v0') # OpenAI Gym의 CartPole 환경 생성

    in\_dim = env.observation\_space.shape[0] # 관찰 공간의 차원 (4차원)

    out\_dim = env.action\_space.n # 행동 공간의 개수 (2가지: 좌/우)

    pi = Pi(in\_dim, out\_dim) # 정책 네트워크 생성

    optimizer = optim.Adam(pi.parameters(), lr=0.01) # Adam 최적화 알고리즘 설정

    for epi in range(300): # 총 300 에피소드 실행

        state = env.reset() # 환경 초기화 및 초기 상태 반환

        for t in range(200): # 최대 타임스텝 200까지 진행

            action = pi.act(state) # 정책에 따라 행동 선택

            state, reward, done, \_ = env.step(action) # 선택한 행동을 환경에 적용

            pi.rewards.append(reward) # 보상 저장

            env.render() # 환경 렌더링 (시각적 출력)

            if done: # 에피소드 종료 조건

                break

        loss = train(pi, optimizer) # 에피소드마다 학습 수행 (REINFORCE)

        total\_reward = sum(pi.rewards) # 에피소드 동안 받은 총 보상 계산

        solved = total\_reward > 195.0 # 보상이 195 이상이면 문제 해결로 간주

        pi.onpolicy\_reset() # 정책의 메모리 초기화 (on-policy)

        # 에피소드 진행 상황 출력

        print(f'Episode {epi}, loss: {loss}, total\_reward: {total\_reward}, solved: {solved}')

# 메인 함수 실행 (스크립트의 시작 지점)

if \_\_name\_\_ == '\_\_main\_\_':

    main()

**Code 2.10 코드 분석**

{

  "reinforce\_cartpole": {

    "agent": [{

      "name": "Reinforce",  // 에이전트 이름 (Reinforce 알고리즘 사용)

      "algorithm": {

        "name": "Reinforce",  // 알고리즘 이름 (Reinforce)

        "action\_pdtype": "default",  // 기본 확률 분포 타입 사용

        "action\_policy": "default",  // 기본 행동 정책 사용

        "center\_return": true,  // 리턴 값 중심화 여부

        "explore\_var\_spec": null,  // 탐색 변수 스펙 (탐색 전략 미설정)

        "gamma": 0.99,  // 할인 계수 (미래 보상에 대한 중요도)

        "entropy\_coef\_spec": {  // 엔트로피 계수에 대한 사양 (탐색-활용 균형 조절)

          "name": "linear\_decay",  // 선형 감소 방식

          "start\_val": 0.01,  // 초기 엔트로피 계수 값

          "end\_val": 0.001,  // 최종 엔트로피 계수 값

          "start\_step": 0,  // 감소 시작 단계

          "end\_step": 20000  // 감소 종료 단계

        },

        "training\_frequency": 1  // 훈련 빈도 (1 스텝마다 훈련)

      },

      "memory": {

        "name": "OnPolicyReplay"  // 메모리 유형 (on-policy 리플레이 메모리)

      },

      "net": {

        "type": "MLPNet",  // 다층 퍼셉트론 네트워크 사용

        "hid\_layers": [64],  // 은닉층 크기 (1개의 은닉층, 64 유닛)

        "hid\_layers\_activation": "selu",  // SELU 활성화 함수

        "clip\_grad\_val": null,  // 그래디언트 클리핑 비활성화

        "loss\_spec": {

          "name": "MSELoss"  // 손실 함수로 MSE (평균 제곱 오차) 사용

        },

        "optim\_spec": {

          "name": "Adam",  // 최적화 알고리즘으로 Adam 사용

          "lr": 0.002  // 학습률 (learning rate)

        },

        "lr\_scheduler\_spec": null  // 학습률 스케줄러 비활성화

      }

    }],

    "env": [{

      "name": "CartPole-v0",  // 환경 이름 (CartPole-v0)

      "max\_t": null,  // 최대 타임스텝 없음

      "max\_frame": 100000  // 최대 프레임 제한 (100,000 프레임)

    }],

    "body": {

      "product": "outer",  // 외부 결합 방식

      "num": 1  // 에이전트의 수 (1명)

    },

    "meta": {

      "distributed": false,  // 분산 학습 비활성화

      "eval\_frequency": 2000,  // 평가 빈도 (2000 스텝마다 평가)

      "max\_session": 4,  // 최대 세션 수 (최대 4회 실행)

      "max\_trial": 1  // 최대 실험 수 (1회)

    },

    "search": {

      "agent": [{

        "algorithm": {

          "gamma\_\_grid\_search": [0.1, 0.5, 0.7, 0.8, 0.90, 0.99, 0.999]  // 감마에 대한 그리드 탐색 (7가지 값)

        }

      }]

    }

  },

  "reinforce\_baseline\_cartpole": {

    "agent": [{

      "name": "Reinforce",  // 에이전트 이름 (Reinforce 알고리즘 사용)

      "algorithm": {

        "name": "Reinforce",  // 알고리즘 이름 (Reinforce)

        "action\_pdtype": "default",  // 기본 확률 분포 타입 사용

        "action\_policy": "default",  // 기본 행동 정책 사용

        "center\_return": true,  // 리턴 값 중심화 여부

        "explore\_var\_spec": null,  // 탐색 변수 스펙 (탐색 전략 미설정)

        "gamma": 0.99,  // 할인 계수 (미래 보상에 대한 중요도)

        "entropy\_coef\_spec": {  // 엔트로피 계수에 대한 사양 (탐색-활용 균형 조절)

          "name": "linear\_decay",  // 선형 감소 방식

          "start\_val": 0.01,  // 초기 엔트로피 계수 값

          "end\_val": 0.001,  // 최종 엔트로피 계수 값

          "start\_step": 0,  // 감소 시작 단계

          "end\_step": 20000  // 감소 종료 단계

        },

        "training\_frequency": 1  // 훈련 빈도 (1 스텝마다 훈련)

      },

      "memory": {

        "name": "OnPolicyReplay"  // 메모리 유형 (on-policy 리플레이 메모리)

      },

      "net": {

        "type": "MLPNet",  // 다층 퍼셉트론 네트워크 사용

        "hid\_layers": [64],  // 은닉층 크기 (1개의 은닉층, 64 유닛)

        "hid\_layers\_activation": "selu",  // SELU 활성화 함수

        "clip\_grad\_val": null,  // 그래디언트 클리핑 비활성화

        "loss\_spec": {

          "name": "MSELoss"  // 손실 함수로 MSE (평균 제곱 오차) 사용

        },

        "optim\_spec": {

          "name": "Adam",  // 최적화 알고리즘으로 Adam 사용

          "lr": 0.002  // 학습률 (learning rate)

        },

        "lr\_scheduler\_spec": null  // 학습률 스케줄러 비활성화

      }

    }],

    "env": [{

      "name": "CartPole-v0",  // 환경 이름 (CartPole-v0)

      "max\_t": null,  // 최대 타임스텝 없음

      "max\_frame": 100000  // 최대 프레임 제한 (100,000 프레임)

    }],

    "body": {

      "product": "outer",  // 외부 결합 방식

      "num": 1  // 에이전트의 수 (1명)

    },

    "meta": {

      "distributed": false,  // 분산 학습 비활성화

      "eval\_frequency": 2000,  // 평가 빈도 (2000 스텝마다 평가)

      "max\_session": 4,  // 최대 세션 수 (최대 4회 실행)

      "max\_trial": 1  // 최대 실험 수 (1회)

    },

    "search": {

      "agent": [{

        "algorithm": {

          "center\_return\_\_grid\_search": [true, false]  // center\_return에 대한 그리드 탐색 (true 또는 false)

        }

      }]

    }

  }

}

Train 명령어 실행 후 그래프

python run\_lab.py slm\_lab/spec/benchmark/reinforce/reinforce\_cartpole.json reinforce\_cartpole train

텍스트, 그래프, 스크린샷, 도표이(가) 표시된 사진

자동 생성된 설명

**첫 번째 그래프: mean\_returns vs. frames**

* **x축 (frames)**: 학습이 진행된 프레임 수를 나타냅니다.
* **y축 (mean\_returns)**: 각 에피소드에서 에이전트가 얻은 평균 리턴(보상)을 나타냅니다.
* **의미**: 에이전트의 성능이 프레임 수에 따라 어떻게 변화하는지를 보여줍니다.
  + 그래프에서 초반에 평균 리턴 값이 낮게 시작하지만, 학습이 진행됨에 따라 점진적으로 증가하여 안정적인 상태에 도달합니다.
  + 음영으로 표시된 영역은 데이터의 분산(표준편차 또는 신뢰구간)을 나타냅니다.
* **평가**:
  + 약 20,000 프레임 이후 평균 리턴이 급격히 상승하고 60,000 프레임 이후 약 200에 근접하며 안정화됩니다.
  + 이는 에이전트가 환경을 점점 잘 학습하여 높은 성능을 달성했음을 의미합니다.

텍스트, 그래프, 스크린샷, 라인이(가) 표시된 사진

자동 생성된 설명

**두 번째 그래프: mean\_returns\_ma vs. frames**

* **x축 (frames)**: 학습이 진행된 프레임 수를 나타냅니다.
* **y축 (mean\_returns\_ma)**: 이동 평균(Moving Average)으로 스무딩 처리된 평균 리턴입니다.
* **의미**: 성능 변화의 전반적인 추세를 더 부드럽고 명확하게 확인하기 위한 그래프입니다.
  + 이동 평균을 사용했기 때문에 단기적인 변동(잡음)이 줄어들어, 장기적인 성능 변화를 더 쉽게 해석할 수 있습니다.
  + 첫 번째 그래프보다 더 부드럽고 일관된 성능 향상이 나타납니다.
* **평가**:
  + 약 20,000 프레임 이후 꾸준히 상승하며, 마지막에는 약 180에 근접한 값으로 수렴합니다.
  + 이는 첫 번째 그래프의 데이터를 기반으로 더 명확하게 성능 개선 추세를 보여줍니다.
* **Reproducing the results in Code 2.12, Code 2.13, and Figure 2.3**

**▪ Draw graphs in Figure 2.3 (textbook) and discussing the results**

**Code 2.12 코드 분석**

{

  "reinforce\_cartpole": {  // 강화학습 환경 설정의 최상위 이름 (Reinforce 알고리즘을 CartPole 환경에서 학습하기 위한 설정)

    "meta": {  // 전반적인 실험 설정을 관리하는 메타데이터

      "distributed": false,  // 분산 학습 사용 여부 (false는 단일 머신에서 학습 실행)

      "eval\_frequency": 2000,  // 평가를 수행하는 빈도 (2000 프레임마다 평가 수행)

      "max\_session": 4,  // 하나의 trial에서 수행할 최대 세션 수

      "max\_trial": 1  // 전체 실험에서 수행할 최대 trial 수

    },

    "search": {  // 하이퍼파라미터 검색을 위한 설정

      "agent": [{  // 에이전트의 하이퍼파라미터를 정의하는 섹션

        "algorithm": {  // 알고리즘 관련 하이퍼파라미터 설정

          "gamma\_\_grid\_search": [0.1, 0.5, 0.7, 0.8, 0.90, 0.99, 0.999]  // 감가율(gamma)의 grid search 범위 설정 (여러 값에 대해 실험 실행)

        }

      }]

    }

  }

}

**Code 2.13 코드 분석**

명령어 실행

python run\_lab.py slm\_lab/spec/benchmark/reinforce/reinforce\_cartpole.json reinforce\_cartpole search

이 설정은 여러 Trial(실험 단위)을 생성하는 Experiment(실험)을 실행합니다. 각 Trial은 원래의 REINFORCE 스펙에 서로 다른 gamma 값을 대입하여 실행됩니다.

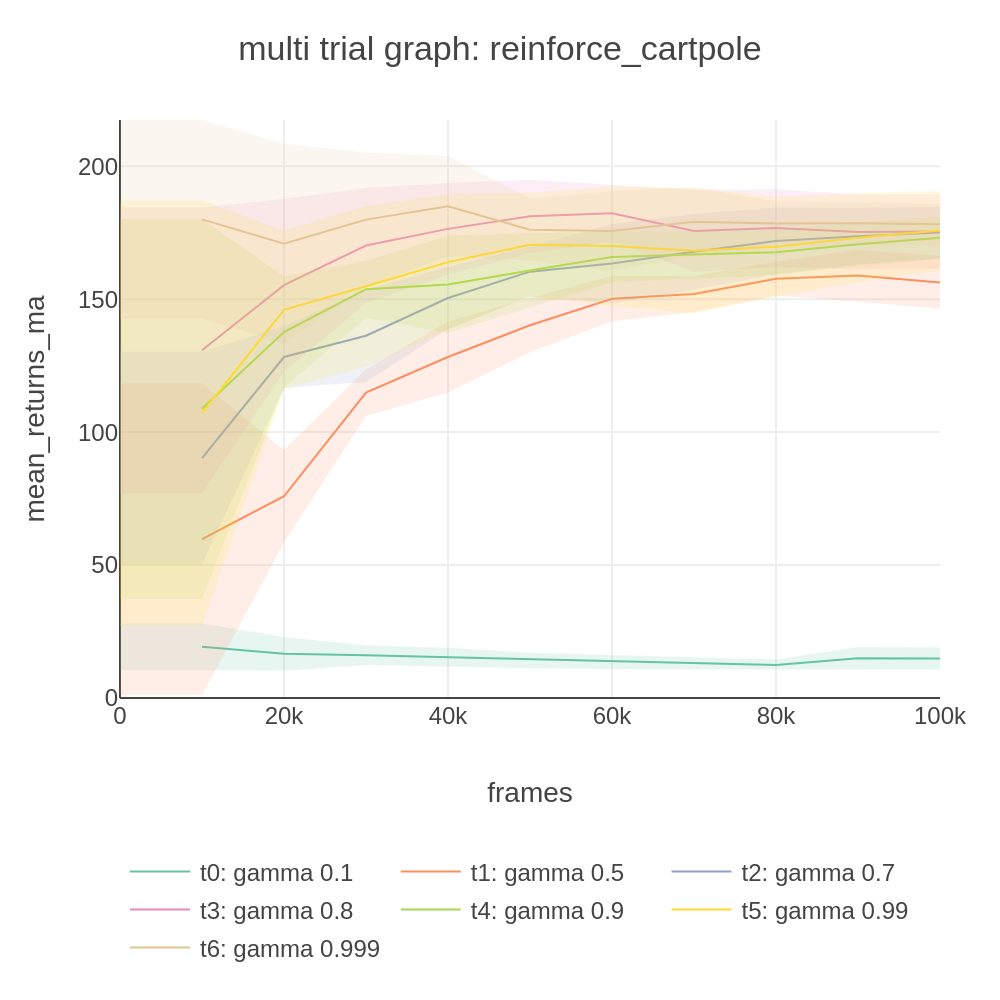
각 Trial은 4번의 반복된 Session(세션)을 실행하여 평균과 함께 오류 대역(error band)을 계산합니다. 이를 바탕으로 멀티 트라이얼(multi-trial) 그래프를 그립니다. 또한, 100개의 평가 지점을 기준으로 이동 평균(moving average)을 적용한 스무딩된 플롯도 제공되며, 이를 통해 비교가 더 용이해집니다.

텍스트, 도표, 스크린샷, 라인이(가) 표시된 사진

자동 생성된 설명

**첫 번째 그래프: mean\_returns vs. frames**

* **x축 (frames)**: 학습이 진행된 프레임 수를 나타냅니다.
* **y축 (mean\_returns)**: 에이전트가 각 에피소드에서 얻은 평균 리턴(보상)을 나타냅니다.
* **선의 의미**: 각 선은 특정 gamma 값에 해당하는 trial의 평균 성능을 나타냅니다.
* **음영(투명 영역)**: 에이전트 성능의 변동성을 나타내며, 보통 표준편차나 신뢰구간을 표시합니다.
* **관찰 결과**:
  + gamma=0.99와 gamma=0.9의 경우 초기 학습이 빠르게 진행되고 높은 평균 리턴 값(200)에 수렴합니다.
  + gamma=0.1과 같은 작은 gamma 값에서는 성능이 거의 향상되지 않습니다. 이는 미래 보상을 거의 고려하지 않기 때문에 학습이 제대로 이루어지지 않은 결과입니다.
  + 중간 값(gamma=0.7, gamma=0.8)에서는 비교적 안정적이지만 최고 성능에 도달하는 속도는 느립니다.



**두 번째 그래프: mean\_returns\_ma vs. frames**

* **x축 (frames)**: 학습이 진행된 프레임 수를 나타냅니다.
* **y축 (mean\_returns\_ma)**: 이동 평균(Moving Average)을 적용한 평균 리턴 값입니다.
* **이동 평균의 역할**: 성능 데이터를 스무딩 처리하여 전반적인 추세를 더 명확히 보여줍니다.
* **관찰 결과**:
  + gamma=0.99와 gamma=0.9는 스무딩된 데이터에서도 일관되게 가장 높은 성능을 보여줍니다.
  + 이동 평균 덕분에 단기적인 변동성(잡음)이 줄어들어 성능 비교가 훨씬 쉬워집니다.
  + 작은 gamma 값(gamma=0.1)에서는 이동 평균을 적용해도 성능이 크게 개선되지 않음을 알 수 있습니다.
* **Reproducing the results in Code 2.14, Code 2.15, and Figure 2.4**

**▪ Draw graphs in Figure 2.4 (textbook) and discussing the results**

**Code 2.14 코드 분석**

{

  "reinforce\_baseline\_cartpole": {  // REINFORCE 알고리즘을 사용한 CartPole 환경에서의 학습 설정

    "meta": {  // 실험 전반에 대한 메타데이터 설정

      "distributed": false,  // 분산 학습 사용 여부 (false: 단일 머신에서 실행)

      "eval\_frequency": 2000,  // 평가를 수행하는 빈도 (2000 프레임마다 평가 실행)

      "max\_session": 4,  // 하나의 trial에서 실행할 최대 세션 수

      "max\_trial": 1  // 전체 실험에서 실행할 최대 trial 수

    },

    "search": {  // 하이퍼파라미터 검색 및 튜닝을 위한 설정

      "agent": [{  // 에이전트 관련 설정

        "algorithm": {  // 알고리즘의 하이퍼파라미터 설정

          "center\_return\_\_grid\_search": [true, false]  // Center return 여부를 결정하는 grid search

          // - true: 보상의 중심화를 활성화

          // - false: 보상의 중심화를 비활성화

        }

      }]

    }

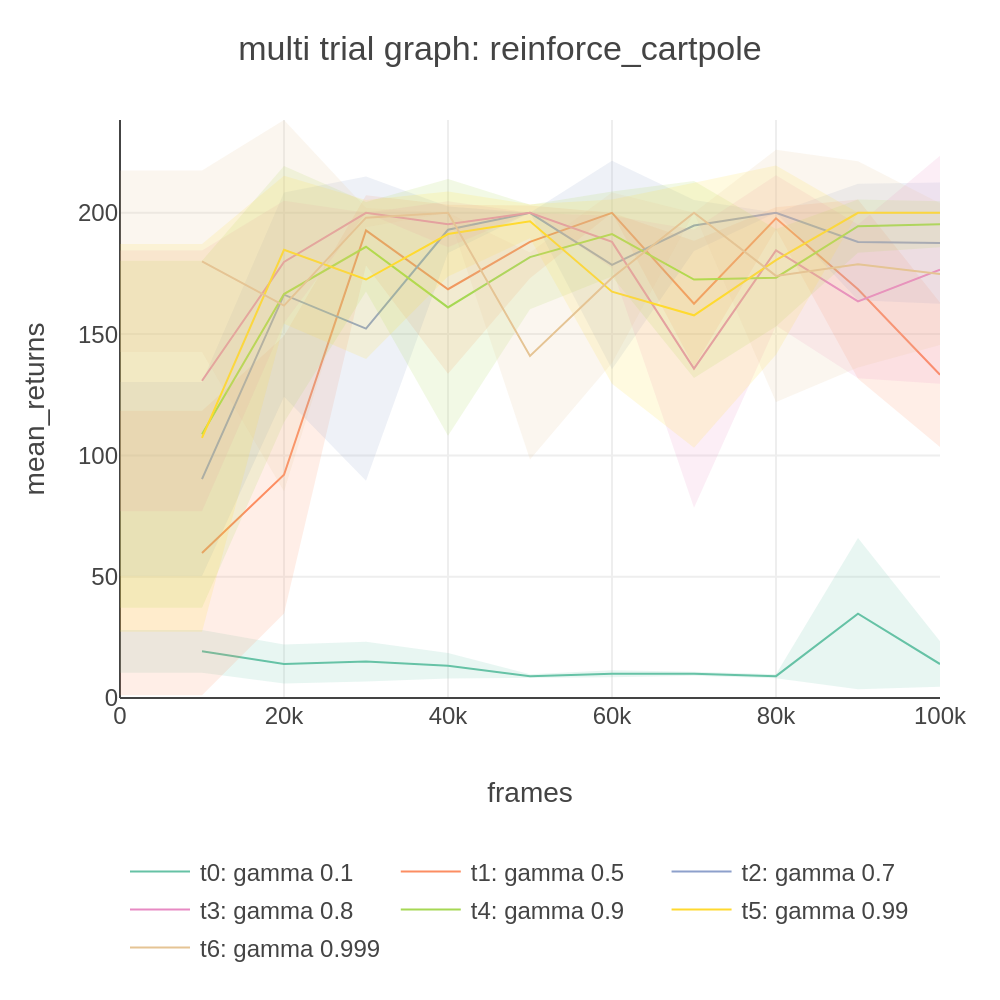
  }

}

**Code 2.15 코드 분석**

python run\_lab.py slm\_lab/spec/benchmark/reinforce/reinforce\_cartpole.json reinforce\_baseline\_cartpole search

스펙 파일에 정의된 대로, 베이스라인을 사용한 경우와 사용하지 않은 경우의 성능을 비교하기 위한 실험을 실행합니다. 이 실험은 두 개의 Trial(실험 단위)과 각 Trial마다 네 개의 Session(세션)으로 구성됩니다. 결과는 Figure 2.4에 표시된 멀티 트라이얼 그래프와 100개의 평가 지점을 기준으로 한 이동 평균 그래프로 나타납니다.

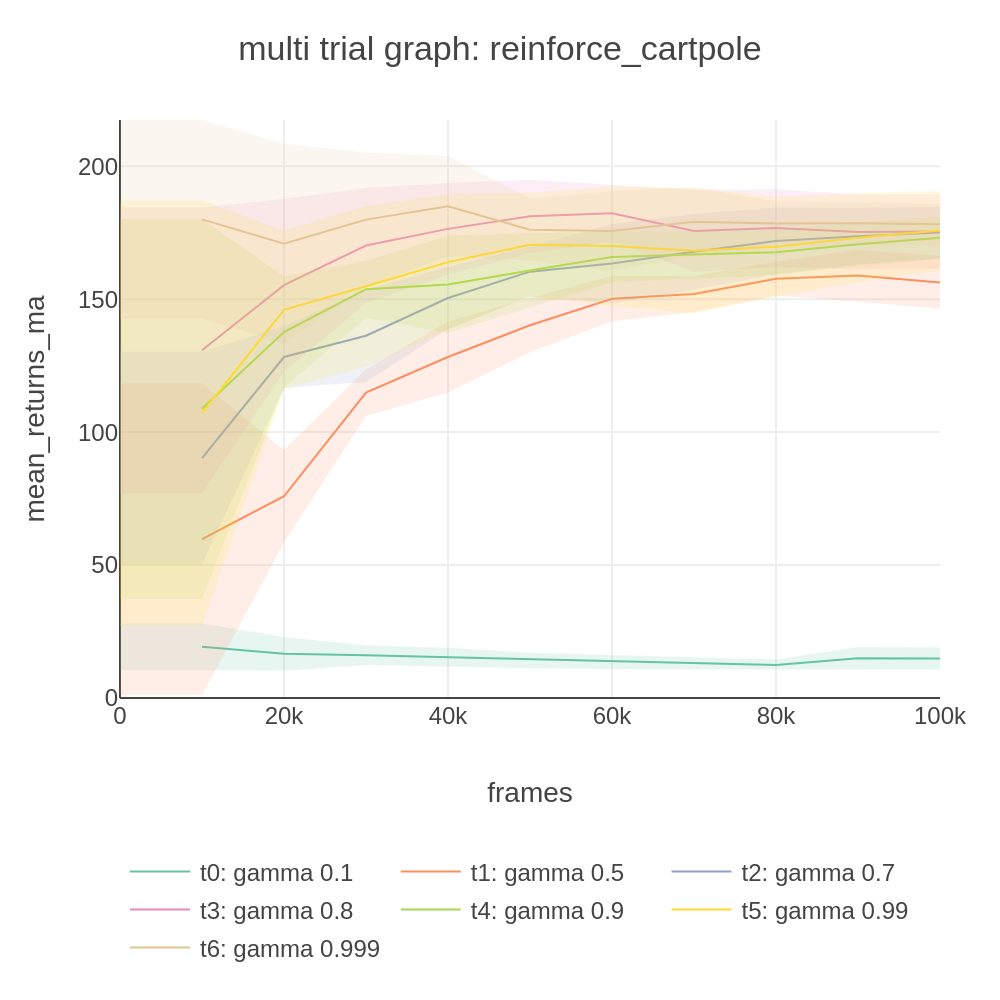


### ****첫 번째 그래프:**** mean\_returns\_ma ****vs.**** frames

* **x축 (frames)**: 학습이 진행된 프레임 수를 나타냅니다.
* **y축 (mean\_returns\_ma)**: 각 평가 지점에서 이동 평균(Moving Average)을 적용한 평균 리턴 값입니다.
* **선의 의미**:
  + 각 선은 특정 gamma 값에 해당하는 trial의 평균 성능을 나타냅니다.
  + 예: t0: gamma=0.1, t5: gamma=0.99 등.

#### ****관찰 결과****:

1. **높은 gamma 값(gamma=0.9, 0.99, 0.999)**:
   * 학습 초기(20,000 프레임 이후)부터 빠르게 성능이 향상되며, 약 40,000~60,000 프레임 이후 안정화됩니다.
   * 최고 평균 리턴 값인 200에 도달하거나 가까운 성능을 보입니다.
   * 이는 장기적인 보상을 중요하게 여기는 높은 gamma 값이 CartPole 환경에서 효과적임을 보여줍니다.
2. **중간 gamma 값(gamma=0.7, 0.8)**:
   * 학습 초기에는 비교적 빠르게 상승하지만, 높은 gamma 값보다 수렴 속도가 느리고 최종 성능도 약간 낮습니다.
3. **낮은 gamma 값(gamma=0.1, 0.5)**:
   * 학습 초기에도 성능이 거의 향상되지 않으며, 최종적으로도 평균 리턴 값이 낮게 유지됩니다.
   * 이는 낮은 gamma 값이 미래 보상을 충분히 고려하지 못하여 학습에 비효율적임을 나타냅니다.



### ****두 번째 그래프:**** mean\_returns ****vs.**** frames

* **x축 (frames)**: 학습이 진행된 프레임 수를 나타냅니다.
* **y축 (mean\_returns)**: 평가 지점에서의 평균 리턴 값입니다.
* **선의 의미**:
  + 각 선은 특정 gamma 값에 해당하는 trial의 성능을 나타냅니다.
  + 첫 번째 그래프와 다르게 이동 평균(Moving Average)을 사용하지 않고 평가 지점에서의 성능을 직접적으로 시각화합니다.

#### ****관찰 결과****:

1. **높은 gamma 값(gamma=0.9, 0.99, 0.999)**:
   * 학습 도중에도 평균 리턴 값이 일정하게 높으며, 성능 변동이 적습니다.
   * 최종적으로 200에 근접한 성능을 보입니다.
2. **중간 gamma 값(gamma=0.7, 0.8)**:
   * 높은 gamma 값에 비해 성능 변동이 크고, 최종 성능도 약간 낮습니다.
3. **낮은 gamma 값(gamma=0.1, 0.5)**:
   * 성능이 전반적으로 낮으며, 변동성은 거의 없으나 최종 결과도 좋지 않습니다.

**결론**

* **gamma=0.99** 또는 **gamma=0.9**가 CartPole 환경에서 REINFORCE 알고리즘을 위한 최적의 감가율로 보입니다.
* 높은 gamma 값은 미래 보상을 더 많이 고려하여 장기적인 보상을 최적화하는 데 효과적입니다.
* 낮은 gamma 값은 단기적인 보상에만 집중하여 학습이 비효율적이거나 성능이 낮습니다.