3η ΑΤΟΜΙΚΗ ΑΣΚΗΣΗ ΤΕΧΝΟΛΟΓΙΕΣ ΕΦΙΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΠΑΝΑΓΙΩΤΗΣ ΠΑΠΑΝΙΚΟΛΑΟΥ

ΑΜ 1067431

PPO

Δεν κατάφερα να κάνω το PPO να μάθει το περιβάλλον.

Για τον κριτή χρησιμοποιήθηκε ένα δίκτυο με την κατάσταση ως είσοδο και την value function της κατάστασης ως επιθυμητή έξοδο και για τον actor δίκτυο με την κατάσταση ως είσοδο και 2 τιμές ως έξοδο, οι οποίες χρησιμοποιούνται ως η μέση τιμή και διασπορά κανονικής κατανομής από την οποία επιλέγεται η δύναμη που θα ασκηθεί όταν το αμάξι βρίσκεται σε εκείνη τη κατάσταση.

Στην αρχή μάθαινε την τακτική να μην κουνιέται καθόλου για να αποφύγει την ποινή και πίστευα ότι έφταιγε μόνο το ότι δεν είχε δει αρκετά παραδείγματα που παίρνει την ανταμοιβή της σημαίας για να μπορέσει να την ψάξει. Δοκίμασα να του φορτώσω αρκετά τυχαία παραδείγματα πριν αρχίσει να δρα ώστε να έχει προηγούμενη γνώση καθώς και να κάνω re-set το πρόγραμμα αν δεν έφτανε την σημαία αρκετά νωρίς ώστε να μάθει για τη σημαία προτού στερεωθεί η στρατηγική ακινησίας.

Αφού αυτά δεν απέδωσαν εφάρμοσα παράλειψη frame ώστε η δύναμη να αλλάζει μόνο κάθε 5 χρονικές στιγμές. Αυτό φαίνεται να διευκόλυνε το πρόβλημα σε σημείο που η τυχαία επιλογή κινήσεων να φτάνει τη σημαία περίπου το 30% των επεισοδίων. Ακόμα και τότε πήγαινε στη τακτική ακινησίας.

Έλεγξα ότι μπορεί να μάθει να σκαρφαλώνει αν δεν του δίνεται ποινή για τα actions. Επίσης δοκίμασα να μειώσω τη ποινή στο 5% της αρχικής. Πρώτα μάθαινε και τότε την ακινησία αλλά αφού μείωσα το learning rate στο 1e-4 κατάφερε μετά από κάποιο σημείο να σκαρφαλώνει στη πλειοψηφία των επεισοδίων. Ακόμα και όταν τα καταφέρνει όμως του παίρνει εκατοντάδες επεισόδια να φτάσει εκεί.

Η βελτίωση λόγω του μειωμένου ρυθμού μάθησης πιστεύω πως οφείλεται στο ότι όταν προσπαθούσε να μειώσει το κόστος του να φτάσει στο στόχο έκανε overcorrect προηγουμένως.

Υπέθεσα πως μειώνοντας το decay θα βελτιωνόταν η απόδοση,

καθώς θα λάμβανε περρισσότερο υπόψη στην αρχή το βραβείο των 100 στο τέλος του επεισοδίου, αλλά δεν παρατήρησα να βοήθησε.

Άλλη αλλαγή που δεν κατέληξε να χρησιμεύσει ήταν να αλλάξω το πόσες φορές ανανεώνονται τα δίκτυα των actor και critic ανά κλήση τής update του αλγορίθμου.

Στην έξοδο του actor χρησιμοποιώ σιγμοειδή συνάρτηση. Από δοκιμές με Relu και tanh φάνηκε να είναι η μόνη που παρήγαγε έστω και λίγο λειτουργικά αποτελέσματα.

Αφού η απόδοση είναι αισθητά χειρότερη της τυχαίας κίνησης είτε κάτι στην υλοποίηση έχει γίνει λάθος με τέτοιο τρόπο που εμποδίζει την εκμάθηση είτε έχω καταλάβει κάποιο βασικό κομμάτι του πως λειτουργούν τα actor-critic / γενικά Q-learning λάθος.

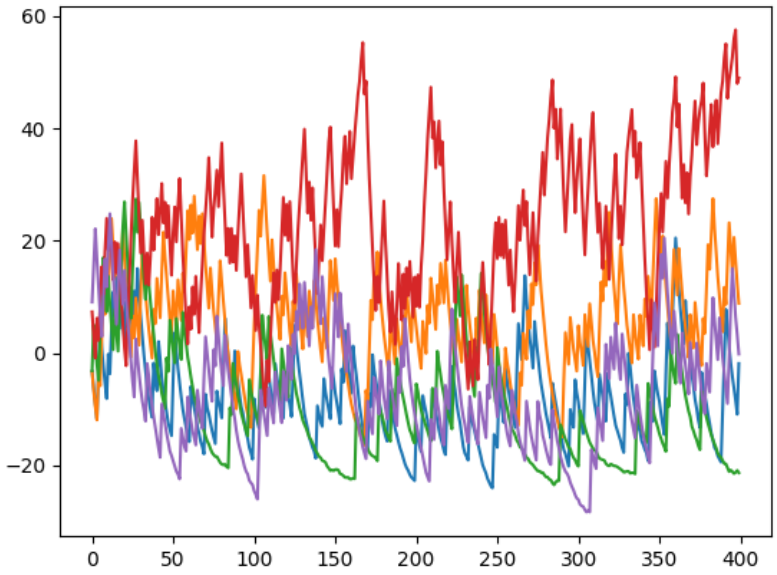
Μετατροπές τις οποίες δεν έχω ελέγξει περιλαμβάνουν την κανονικοποίηση / κωδικοποίηση των εισόδων των δικτύων, καθώς και παραλαγές του περιβάλοντος π.χ. επιστρέφοντας επιβράβευση για το πόσο ψηλά πήγε το αμάξι.

TD3

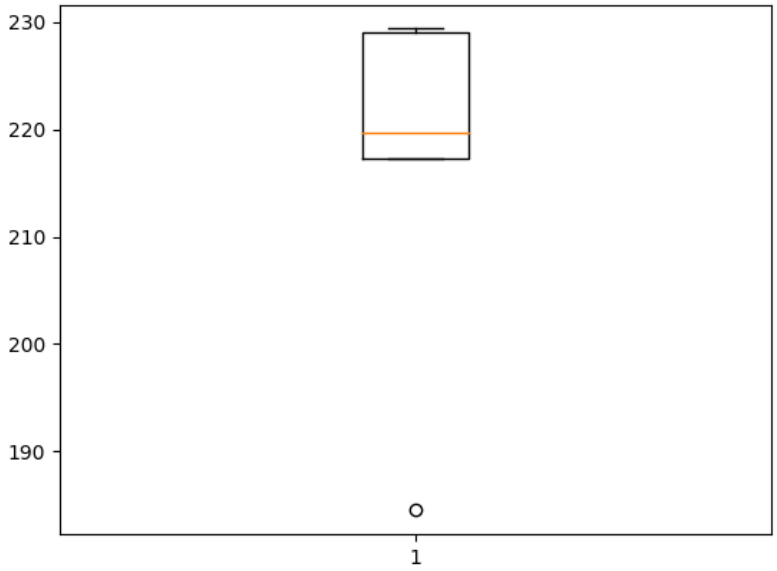
Ό TD3 δεν έχει υλοποιηθεί

DIAGRAMS

- Expected Return



- Execution Times



PPO IMPLEMENTATION

```
from machin.frame.algorithms import PPO
from machin.utils.logging import default_logger as logger
from torch.distributions import Normal, Uniform
import torch as t
import torch.nn as nn
import gym
import matplotlib.pyplot as plt
import time

# configurations
env = gym.make("MountainCarContinuous-v0")
observe_dim = 2
max_episodes = 400
max_steps = 999
solved_repeat = 3


# model definition
class Actor(nn.Module):
    def __init__(self, state_dim):
        super().__init__()

        self.fc1 = nn.Linear(state_dim, 16)
        self.fc2 = nn.Linear(16, 16)
        self.fc3 = nn.Linear(16, 2)

    def forward(self, state, action=None):
        a = t.relu(self.fc1(state))
        a = t.relu(self.fc2(a))
        probs = t.sigmoid(self.fc3(a))
        mu = probs[0][0] * 2 - 1
        sig = probs[0][1]
        dist = Normal(mu, sig)
        act = action if action is not None else dist.sample()
        act_entropy = dist.entropy()
        act_log_prob = dist.log_prob(act.flatten())
        return act, act_log_prob, act_entropy


class Critic(nn.Module):
    def __init__(self, state_dim):
        super().__init__()
```

```
        self.fc1 = nn.Linear(state_dim, 16)
        self.fc2 = nn.Linear(16, 16)
        self.fc3 = nn.Linear(16, 1)

    def forward(self, state):
        v = t.relu(self.fc1(state))
        v = t.relu(self.fc2(v))
        v = self.fc3(v)
        return v


def train():
    actor = Actor(observe_dim)
    critic = Critic(observe_dim)

    ppo = PPO(
        actor, critic,
        t.optim.Adam, nn.MSELoss(reduction="sum"),
        actor_update_times=5, critic_update_times=10,
        actor_learning_rate=1e-3, critic_learning_rate=1e-3,
        discount=0.99)

    episode, step, wins = 0, 0, 0
    smoothed_total_reward = 0
    Scores = []
    Dist = []

    while episode < max_episodes:
        episode += 1
        total_reward = 0
        terminal = False
        step = 0
        state = t.tensor(env.reset(), dtype=t.float32).view(1, observe_dim)
        changeRate = 5
        unif = Uniform(-1, 1)
        maxDist = -10
        tmp_observations = []
        while not terminal and step <= max_steps:
            # if episode % 100 == 0:
            # env.render()
            step += 1
            with t.no_grad():
                old_state = state
                # agent model inference
                if step % changeRate == 1:
                    if episode > 30:
```

```
                    action = ppo.act({"state": old_state})[0]
                else:
                    action = unif.sample()
            state, reward, terminal, _ = env.step([action])
            state = t.tensor(state, dtype=t.float32).view(1, observe_dim)
            # reward = max(reward * 0.05, reward)
            total_reward += reward
            maxDist = max(maxDist, state[0][0])

            tmp_observations.append(
                {
                    "state": {"state": old_state},
                    "action": {"action": t.tensor([[action]])},
                    "next_state": {"state": state},
                    "reward": reward,
                    "terminal": terminal or step == max_steps,
                }
            )

        # update
        ppo.store_episode(tmp_observations)
        if total_reward > 0:
            wins += 1
            logger.info("On Flag")

        # show reward
        smoothed_total_reward *= 0.9
        smoothed_total_reward += total_reward * 0.1
        winrate = wins / max(episode, 1)
        Info = (f"Ep {episode} >={maxDist:.2f} "
                f"$={smoothed_total_reward:.2f} "
                f"% {winrate:.2f}")
        logger.info(Info)
        Scores.append(smoothed_total_reward)
        Dist.append(maxDist)
    return Scores


if __name__ == "__main__":
    plt.figure(1)
    durations = []
    for _ in range(5):
        a = time.time()
        x = train()
        durations.append(time.time() - a)
        plt.plot(x)
```

```
plt.savefig("scores.png")
plt.figure(2)
plt.boxplot(durations)
plt.savefig("times.png")
```