Deep Reinforcement Learning

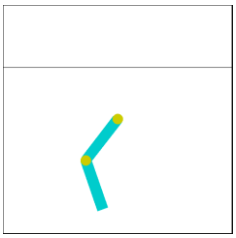
ASSIGNMENT 2

Dor Shmuel - 312179971 | Yuval Cohen - 314978198 | 14/12/2022

Section 1 – Training individual networks

In this section, we will train individual agent models based on the actor-critic algorithm for each of the selected environments.

1.1 Actor-Critic algorithm for "Acrobot-v1"



Description: two blue links connected by two green joints. The joint between the two links is actuated. The goal is to swing the free end of the outer link to reach the target height (black horizontal line above the system) by applying torque on the actuator. [1]

Action space shape: Discrete (3).

Observation shape: (6,).

Architectures

1. The architecture of the actor – **Policy Network**:

   Input: dense layer, **size**: 6

   Hidden 1: dense layer, **size**: 64

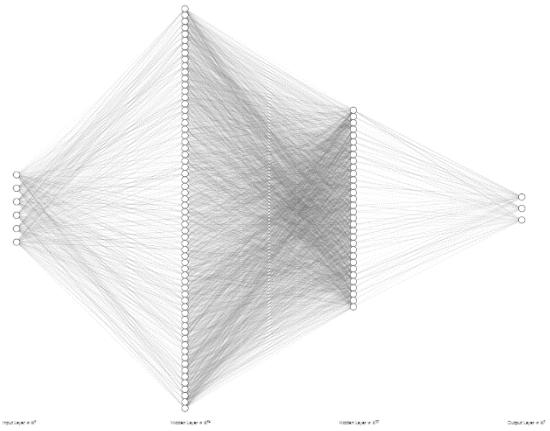
   Activation: ReLU

   Hidden 2: dense layer, **size**: 32

   Activation: ReLU

   Output: dense layer, **size**: 3

   Activation: Linear



2. **The a**rchitecture of the critic – Value Network:

   Input: dense layer**, size:** 6

   Hidden 1: dense layer**, size:** 64

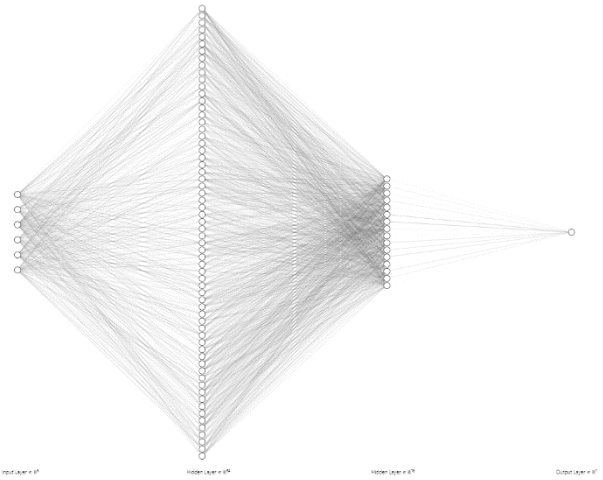
   Activation: ReLU

   Hidden 2: dense layer**, size:** 16

   Activation: ReLU

   Output: dense layer**, size:** 3

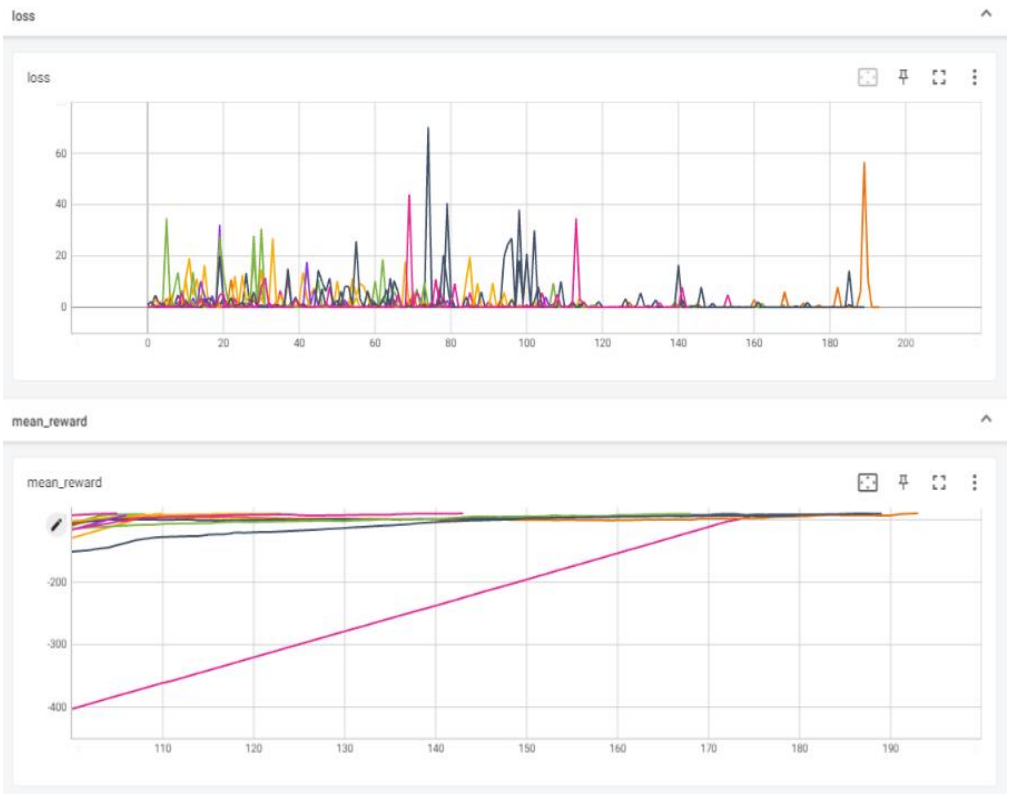
   Activation: Linear



Hyper-parameters Tuning

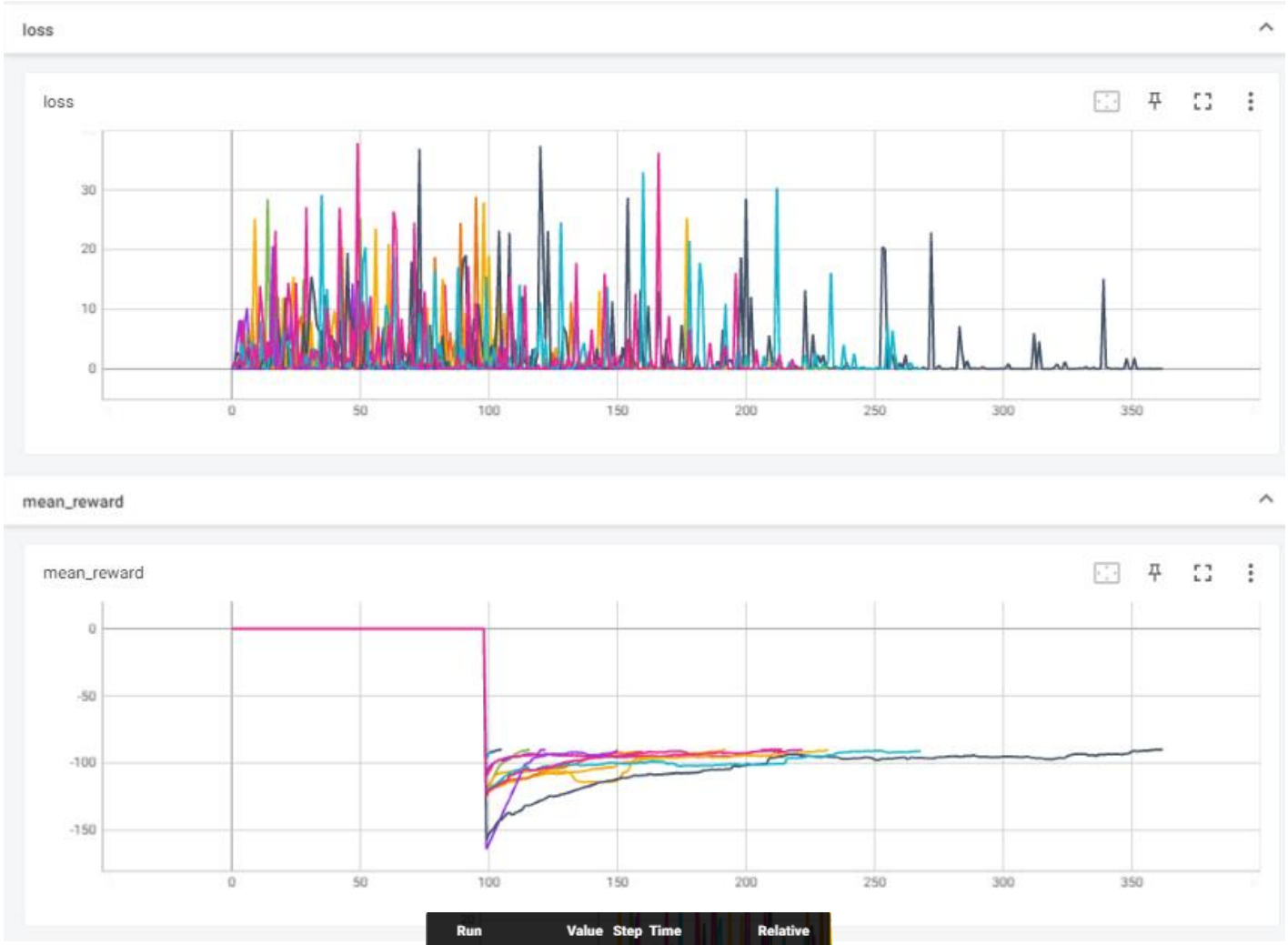
The actor-critic algorithm is implemented over the discussed environment to reach a mean reward of -90 points for 100 consecutive episodes(documentation goal: -100).
We set the same seed for optimal tuning through all the tuning procedures.

1. **learning rate of critic-value network tuning**:
   reference values for tuning: actor learning rate: 0.0005, discount factor: 0.99.
   for learning rate **lr=0.002** the agent converged into the desired goal after **105** episodes.



Run	Value	Step	Time	Relative
sv_lr=0.0007	-90	193	1/7/23, 5:31 PM	473.6 ms
sv_lr=0.002	-90	105	1/7/23, 5:31 PM	259.2 ms
sv_lr=0.01	-90.01	143	1/7/23, 5:31 PM	362.3 ms
sv_lr=0.007	-90.22	189	1/7/23, 5:31 PM	472.8 ms
sv_lr=0.003	-90.36	123	1/7/23, 5:31 PM	307.2 ms
sv_lr=0.0004	-90.6	110	1/7/23, 5:31 PM	276.9 ms
sv_lr=0.005	-90.92	168	1/7/23, 5:31 PM	414.3 ms
sv_lr=0.0009	-90.99	188	1/7/23, 5:31 PM	463 ms
sv_lr=0.0006	-91.03	108	1/7/23, 5:31 PM	277.3 ms
sv_lr=0.0003	-91.2	176	1/7/23, 5:31 PM	441.3 ms
sv_lr=0.004	-92.83	106	1/7/23, 5:31 PM	259.5 ms

2. **learning rate of actor-policy tuning**:
   reference values for tuning: critic learning rate: 0.002, discount factor: 0.99.
   for learning rate **lr=0.0006** the agent converged into the desired goal after **101** episodes.



Run	Value	Step	Time	Relative
policy_lr=0.0005	-90	105	1/7/23, 7:12 PM	309.2 ms
policy_lr=0.0007	-90.08	214	1/7/23, 7:12 PM	598.8 ms
policy_lr=0.0002	-90.1	122	1/7/23, 7:12 PM	342.9 ms
policy_lr=5e-05	-90.1	362	1/7/23, 7:12 PM	1.047 sec
policy_lr=0.0003	-90.2	116	1/7/23, 7:12 PM	328.4 ms
policy_lr=9e-05	-90.21	222	1/7/23, 7:12 PM	627 ms
policy_lr=0.0009	-90.24	232	1/7/23, 7:12 PM	653.6 ms
policy_lr=0.001	-90.27	150	1/7/23, 7:12 PM	426.3 ms
policy_lr=0.0001	-90.3	192	1/7/23, 7:12 PM	564.6 ms
policy_lr=0.0006	-90.71	101	1/7/23, 7:12 PM	290.7 ms
policy_lr=7e-05	-91.11	268	1/7/23, 7:12 PM	763.8 ms
policy_lr=0.0004	-91.33	160	1/7/23, 7:12 PM	448.7 ms

Optimal hyper-parameters:

Learning rate – policy (actor)	Learning rate – value (critic)	Discount factor
0.0006	0.002	0.99

Time for convergence:

Solved at episode: **102**, Algorithm actor_critic for Acrobot-v1 converged after **20.437** seconds (Trained over GPU: Nvidia RTX 3070 8Gb)

In the manner of **performances**, we can notice that the actor-critic algorithm converges very fast, allowing us to properly learn how to move the tip of the acrobat over the height line.

Running the script:
To run the above simulation, run the script **actor_critic.py**, and set the following parameters:

a. env_name = 'Acrobot-v1'

1.2 Actor-Critic algorithm for "CartPole-v1"

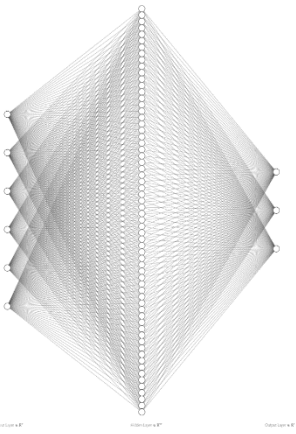
We have already dealt with the "CartPole-v1" environment in assignment 2. This time we increased the size of the hidden layer of the policy network from 12 (as we set in assignment 2) to 64.

Action space shape: Discrete (2).

Observation shape: (4,).

Architectures

The same architecture was built for each of the environments



1. The architecture of the actor – **Policy Network**:

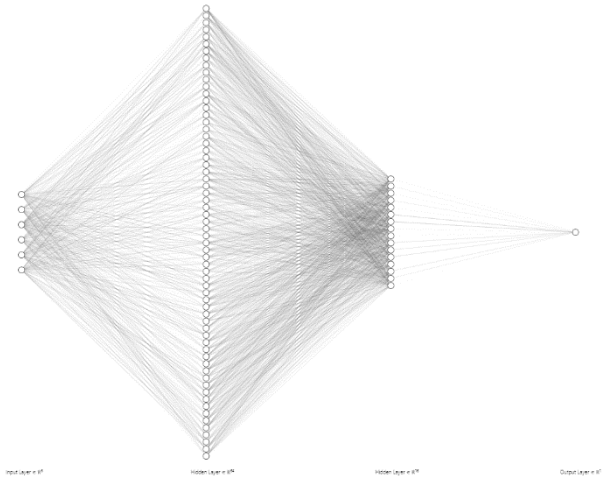
   Input: dense layer, **size**: 6

   Hidden 1: dense layer, **size**: 64

   Activation: ReLU

   Output: dense layer, **size**: 3

   Activation: Linear



2. The architecture of the critic – Value Network:

   Input: dense layer**, size:** 6

   Hidden 1: dense layer**, size:** 64

   Activation: ReLU

   Hidden 2: dense layer**, size:** 16

   Activation: ReLU

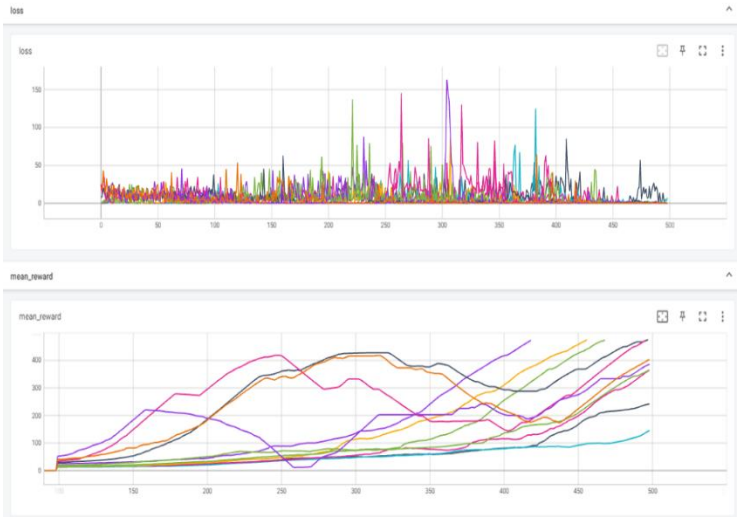
   Output: dense layer**, size:** 3

   Activation: Linear

Hyper-parameters Tuning

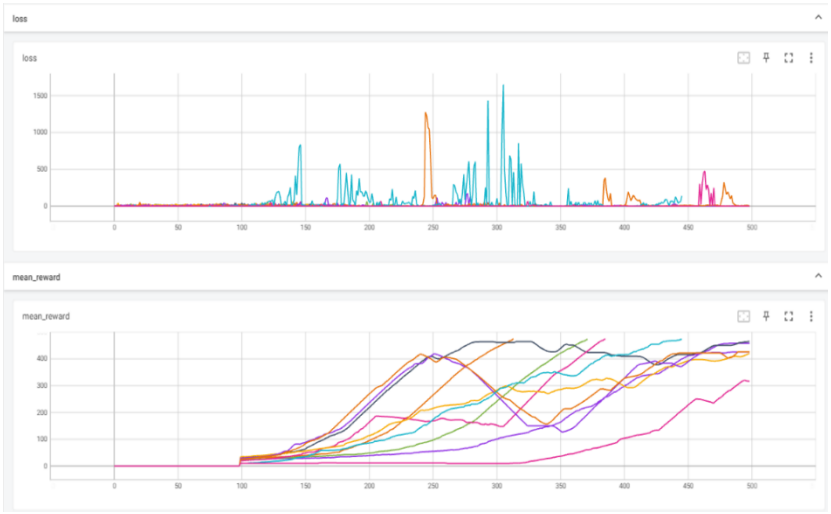
The actor-critic algorithm is implemented over the discussed environment to reach a mean reward of **475** points for 100 consecutive episodes(documentation goal: 475).
We set the same seed for optimal tuning through all the tuning procedures.

1. **learning rate of critic-value network tuning**:
   reference values for tuning: actor learning rate: 5e-5, discount factor: 0.99.
   we can notice that as expected, for the learning rate of **lr=0.0009** the model **successfully** converged into the desired goal after **418** episodes.



Run	Value	Step	Time	Relative
sv_lr=0.0007	474.7	456	1/9/23, 11:11 PM	1.342 sec
sv_lr=0.003	474.6	497	1/9/23, 11:11 PM	1.431 sec
sv_lr=0.001	473.6	468	1/9/23, 11:11 PM	1.366 sec
sv_lr=0.0009	472.4	418	1/9/23, 11:11 PM	1.208 sec
sv_lr=0.005	472.1	495	1/9/23, 11:11 PM	1.425 sec
sv_lr=0.009	403.6	498	1/9/23, 11:11 PM	1.428 sec
sv_lr=0.007	386.1	498	1/9/23, 11:11 PM	1.461 sec
sv_lr=0.0006	364.8	498	1/9/23, 11:11 PM	1.476 sec
sv_lr=0.0008	363.2	498	1/9/23, 11:11 PM	1.458 sec
sv_lr=0.0004	242	498	1/9/23, 11:11 PM	1.437 sec
sv_lr=0.0005	144	498	1/9/23, 11:11 PM	1.436 sec

2. **learning rate of actor-policy tuning**:
   reference values for tuning: critic learning rate: 0.006, discount factor: 0.99.
   for learning rate **lr=0.0003** the agent converged into the desired goal after **313** episodes.



Run	Value	Step	Time	Relative
policy_lr=0.0003	474.3	313	1/10/23, 12:45 AM	812 ms
policy_lr=0.0006	474.3	385	1/10/23, 12:45 AM	996.8 ms
policy_lr=0.0002	473.3	371	1/10/23, 12:45 AM	956.5 ms
policy_lr=0.004	472.9	445	1/10/23, 12:46 AM	1.149 sec
policy_lr=0.0004	465.3	498	1/10/23, 12:45 AM	1.289 sec
policy_lr=0.0001	458.6	498	1/10/23, 12:45 AM	1.295 sec
policy_lr=0.002	426.5	498	1/10/23, 12:46 AM	1.313 sec
policy_lr=0.0009	425.9	498	1/10/23, 12:46 AM	1.309 sec
policy_lr=0.0007	420.8	498	1/10/23, 12:46 AM	1.294 sec
policy_lr=0.005	316.2	498	1/10/23, 12:46 AM	1.277 sec

Optimal hyper-parameters:

Learning rate – policy (actor)	Learning rate – value (critic)	Discount factor
0.0003	0.0009	0.99

Time for convergence:

Solved at episode: 314
Algorithm actor_critic for CartPole-v1 converged after 142.601 seconds

1. In the manner of **performances**, the actor-critic algorithm converges to the desired goal.
2. We can also notice that for more complex architecture (with more elements in the layers) the performances of the model over the Cart Pole environment are degrading, where the architecture which we design in assignment 2, contains more than 4 times less neurons than the current architecture but attains the desired goal after 253 episodes (in comparison to 314) and after 95.3602 seconds, which is only 70% of the time required to our current model to convergence.
   Running the script:
   To run the above simulation, run the script **actor_critic.py**, and set the following parameters:
   a. env_name = Cartpole-v1'

1.3 Actor-Critic algorithm for "MountainCarContinuous-v0"

Description: In this environment, a car is placed at the bottom of a valley. The only possible actions are the continuous value accelerations that can be applied to the car in either direction. The goal is to reach the flag on top of the right hill.

Action space shape: Box(-1.0, 1.0, (1,), float32). i.e. any velocity value in the range of [-1, 1].

Observation shape: (2,) – position and velocity.

Architectures

3. Architecture of the actor – **Policy Network**:

   Input: dense layer, **size**: 6

   Hidden 1: dense layer, **size**: 64

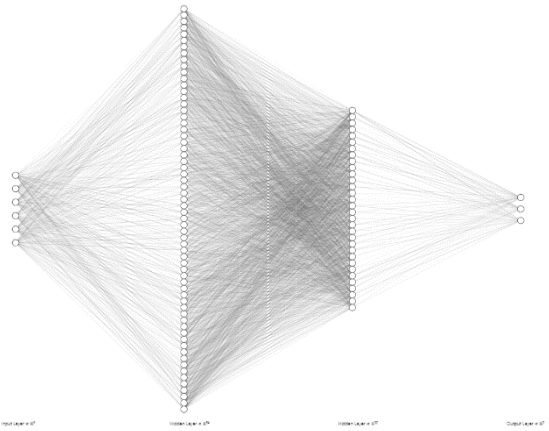
   Activation: ReLU

   Hidden 2: dense layer, **size**: 32

   Activation: ReLU

   Output: dense layer, **size**: 3

   Activation: Linear



4. Architecture of the critic – Value Network:

   Input: dense layer**, size:** 6

   Hidden 1: dense layer**, size:** 64

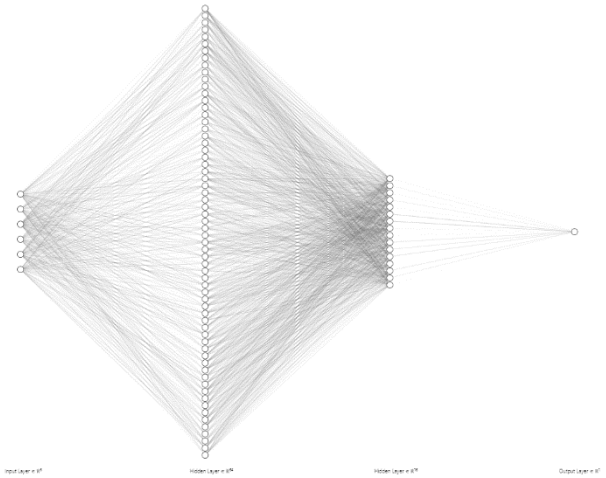
   Activation: ReLU

   Hidden 2: dense layer**, size:** 16

   Activation: ReLU

   Output: dense layer**, size:** 3

   Activation: Linear



Hyper-parameters Tuning

The actor-critic algorithm is implemented over the discussed environment to reach a mean reward of **75** points for 100 consecutive episodes (documentation goal: None).
To generate continued value by the policy network, the output layers contain two nodes, which resemble the expectation (mu) and the standard derivation (sigma) of gaussian probability, from which the action is sampled.

The award function of the environment is as follows: agent gets 100 points when reaching the flag at the top hill, while at each step he doesn't, he is punished by a negative award of -0.1 * $action^2$ points, i.e. higher loss for applying actions with higher velocities.

By running a few simulations over the environment (as we can observe in the plot below), the agent has learned that minor changes in the velocity will yield an optimal overall mean award near ~0 (actions with small values, which results in mean -> 0), instead of trying to reach the flag by applying actions with high velocity, risking with high punishment. This reward function raises an exploration challenge because if the agent does not reach the target soon enough, it will figure out that it is better not to move and won't find the target anymore.

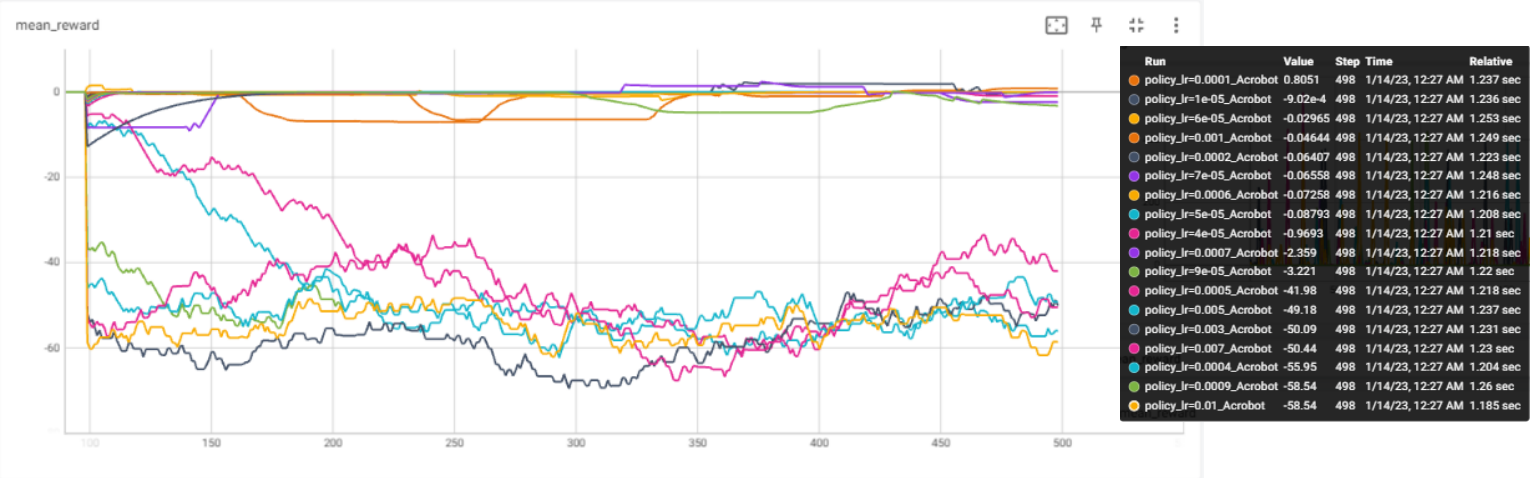


Figure 1 - simulation of MountainCar with an authentic reward function

Hence, the Initial results didn't converge to the desired goal, and some modifications need to be made. We have been trying to cheer the agent with different custom awards.

First, with offered an additional award with relation to the agent **position**:

a. +1 for each step agent is above p=0.05
b. +5 for position = 0.25
c. 500-step for reaching the top.

This modified reward let the agent achieve some positive rewards (for the first time, as we can notice in the plot below) but it still doesn't achieve the goal, and in most of the simulations as steps go by, it converges into almost zero mean awards.

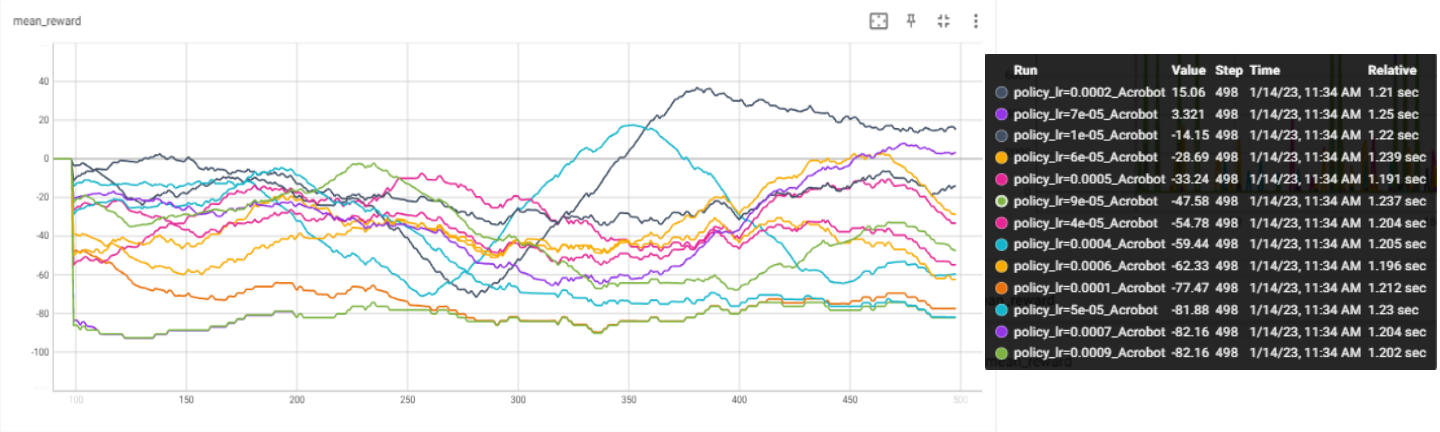


Figure 2 - simulation of MountainCar with a basic modified reward function

As a result, we have figured out that the only way that our agent will get to the top of the hill, is by swinging him back and forth from one side to another, where at each step he is accelerating to the opposite side until he reached the flag. Hence, we would like to motivate him by awarding the agent **each time he highly accelerates**, i.e. additional reward with the difference between consecutive step velocities.

New reward = environment reward + 100 * $factor$* ($|Vel_{next\ state}| - |Vel_{current\ state}|$)

```
reward = reward + 100 * factor * (abs(next_state[1]) - abs(state[0][1]))
```

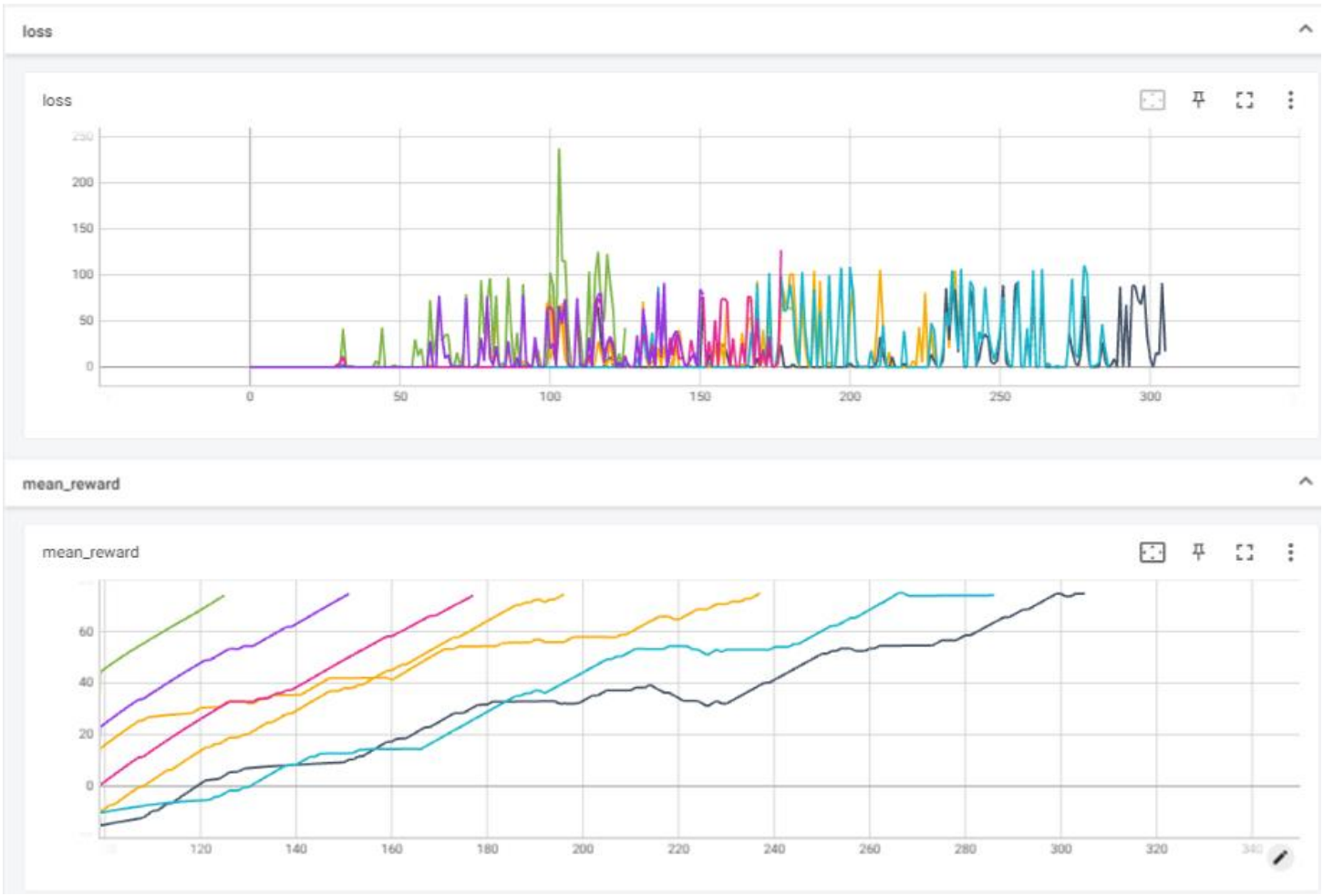
Hence, the agent will be rewarded relative to the velocity difference between adjacent steps (high reward for a high difference in the velocities of adjacent steps).

We set the same seed for optimal tuning through all the tuning procedures.

3. **learning rate of critic-value network tuning**:
   reference values for tuning: actor learning rate: 0.005, discount factor: 0.99.
   we can notice that for the learning rate of **lr=5e-5** the model **successfully** converged into the desired goal after **201** episodes.

Run	Value	Step	Time	Relative
sv_lr=0.0006	74.94	288	1/7/23, 5:42 PM	730.9 ms
sv_lr=5e-05	74.46	196	1/7/23, 5:42 PM	516.8 ms
sv_lr=1e-05	74.45	311	1/7/23, 5:42 PM	867.4 ms
sv_lr=0.005	74.41	400	1/7/23, 5:43 PM	1.064 sec
sv_lr=0.0002	74.36	201	1/7/23, 5:42 PM	535.6 ms
sv_lr=0.0001	74.3	253	1/7/23, 5:42 PM	684.3 ms
sv_lr=0.01	73.88	395	1/7/23, 5:43 PM	1.07 sec

4. **learning rate of actor-policy tuning**:
   reference values for tuning: critic learning rate: 0.006, discount factor: 0.99.
   for learning rate **lr=0.0002** the agent converged into the desired goal after **430** episodes.



Run	Value	Step	Time	Relative
policy_lr=0.0001	74.82	237	1/7/23, 10:18 PM	678.8 ms
policy_lr=4e-05	74.7	305	1/7/23, 10:18 PM	931 ms
policy_lr=9e-05	74.56	151	1/7/23, 10:18 PM	433.9 ms
policy_lr=5e-05	74.46	196	1/7/23, 10:28 PM	513 ms
policy_lr=6e-05	74.03	286	1/7/23, 10:18 PM	884.2 ms
policy_lr=7e-05	74.03	177	1/7/23, 10:18 PM	506.3 ms
policy_lr=0.0002	74.02	125	1/7/23, 10:18 PM	365.1 ms

Optimal hyper-parameters:

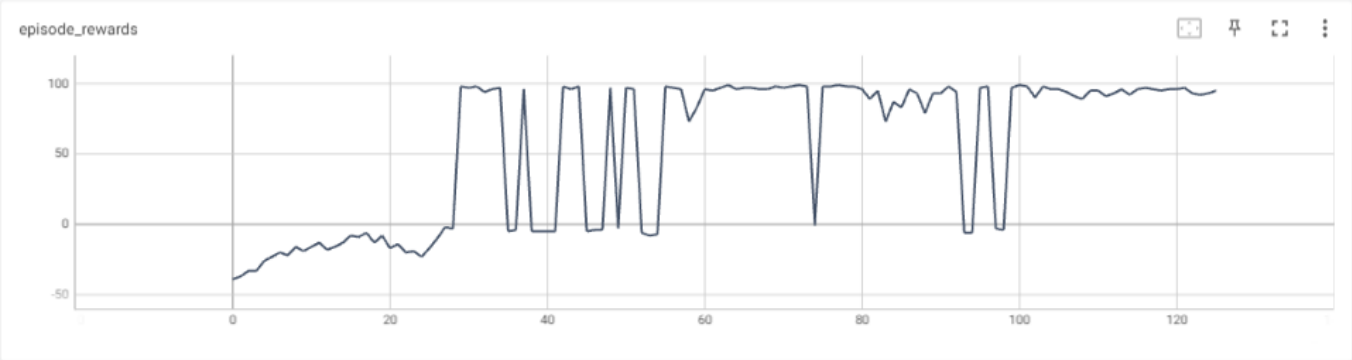
Learning rate – policy (actor)	Learning rate – value (critic)	Discount factor
0.0002	5e-5	0.99

Time for convergence:

Solved at episode: 126, Algorithm actor_critic for MountainCarContinuous-v0 converged after 151.823 seconds

In the manner of **performances**, we can notice that the modified reward has significantly improved the ability of the agent to converge toward the desired goal.
As we wished, the acceleration-aided reward function enables the agent to heavily swing from side to die, reaching the desired goal fast (after 126 episodes, 151 seconds, where most of the final episodes successfully reached the flag after a small number of steps)

Last episodes in the training procedure:



Episode 116, steps 94, Reward: 97.05 Average over 100 episodes: 64.03
Episode 117, steps 91, Reward: 96.99 Average over 100 episodes: 65.07
Episode 118, steps 180, Reward: 95.32 Average over 100 episodes: 66.16
Episode 119, steps 91, Reward: 96.95 Average over 100 episodes: 67.22
Episode 120, steps 104, Reward: 96.22 Average over 100 episodes: 68.35
Episode 121, steps 98, Reward: 97.0 Average over 100 episodes: 69.47
Episode 122, steps 299, Reward: 94.67 Average over 100 episodes: 71.72
Episode 123, steps 559, Reward: 93.17 Average over 100 episodes: 72.89
Episode 124, steps 188, Reward: 95.67 Average over 100 episodes: 74.02
Episode 125, steps 270, Reward: 95.12 Average over 100 episodes: 75.08).

Running the script:
To run the above simulation, run the script **actor_critic_mountain_car.py**

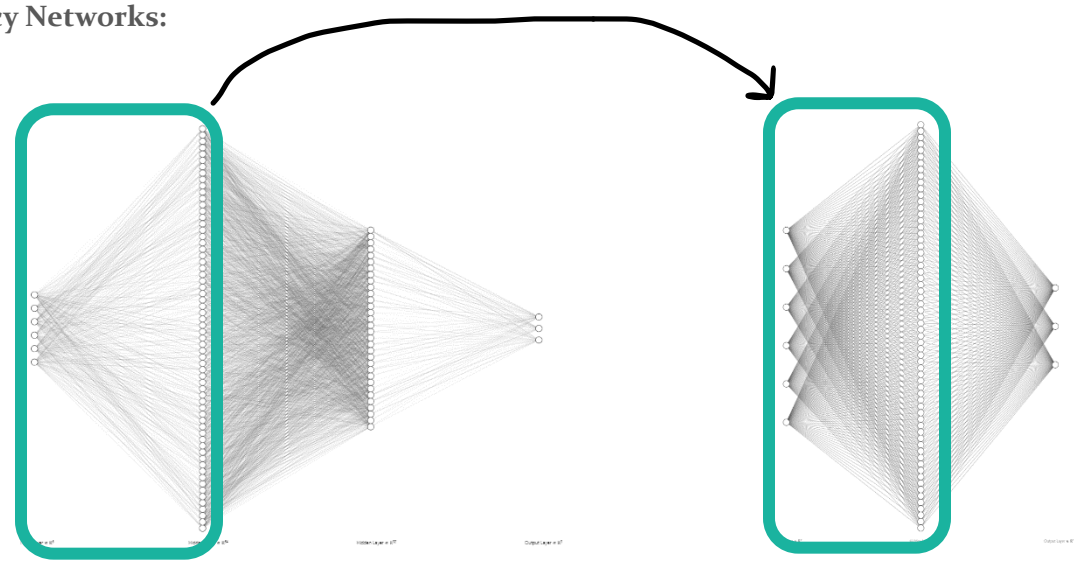
Section 2 – Fine-Tune an existing model

In this section, we will use the fully trained models from the previous section and leverage their success in the training procedure to fine-tune another model. In the fine-tuning procedure will freeze the trained layers except the final one, which will be initialized and re-trained for the new environment.

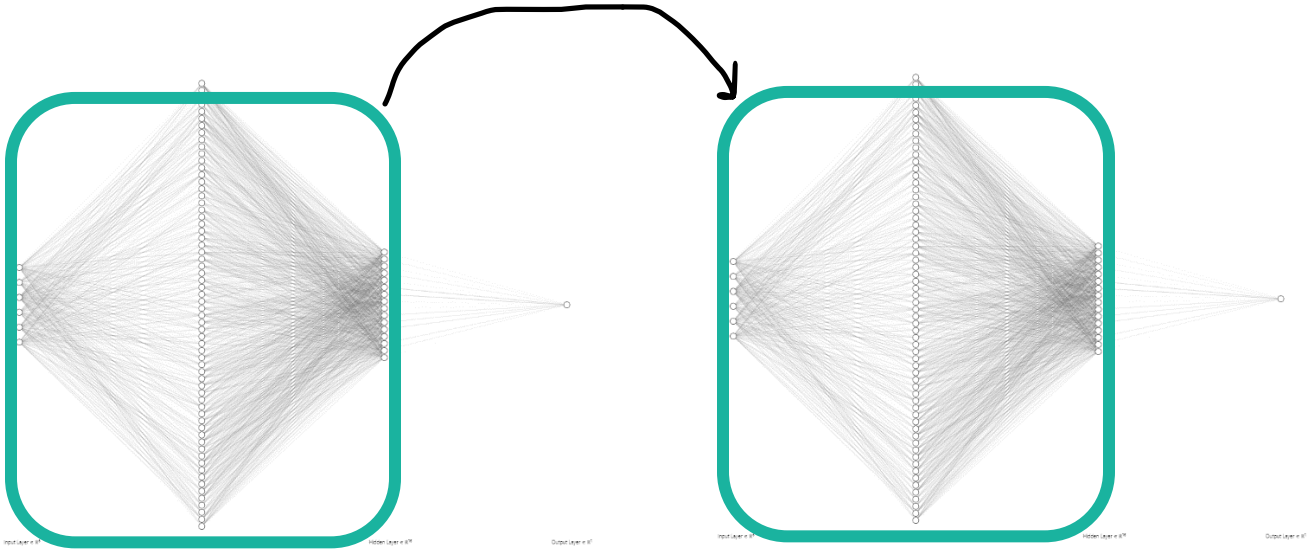
1.4 Fine-tuning of Cartpole environment based on Acrobot-v1 model

In this section, we will observe the architecture that has been trained on the "Acrobot-v1" environment and implant its first hidden layer weights in the architecture that is intended to be trained over the "Cartpole-v1".

Policy Networks:

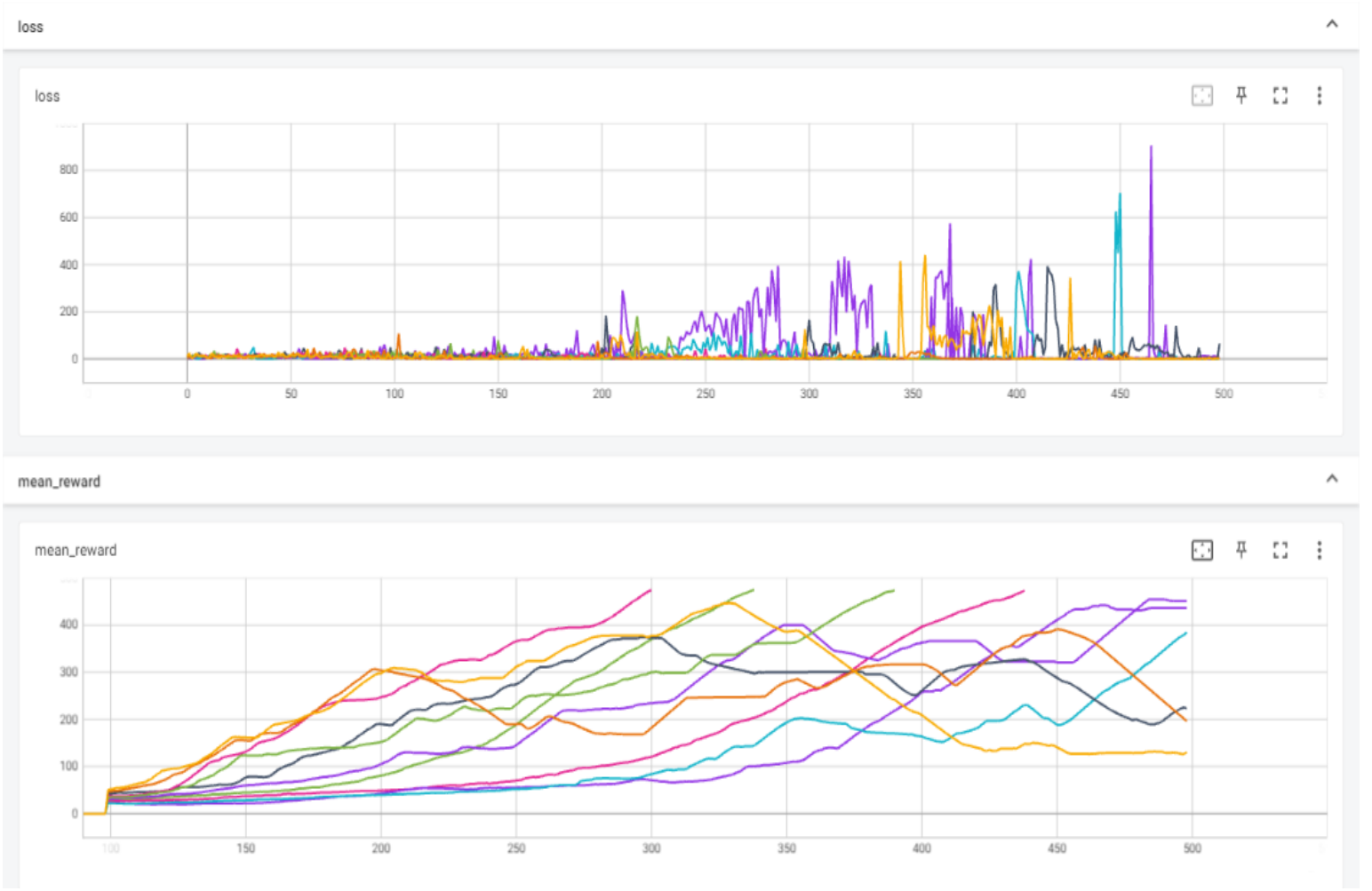


State-Value networks:



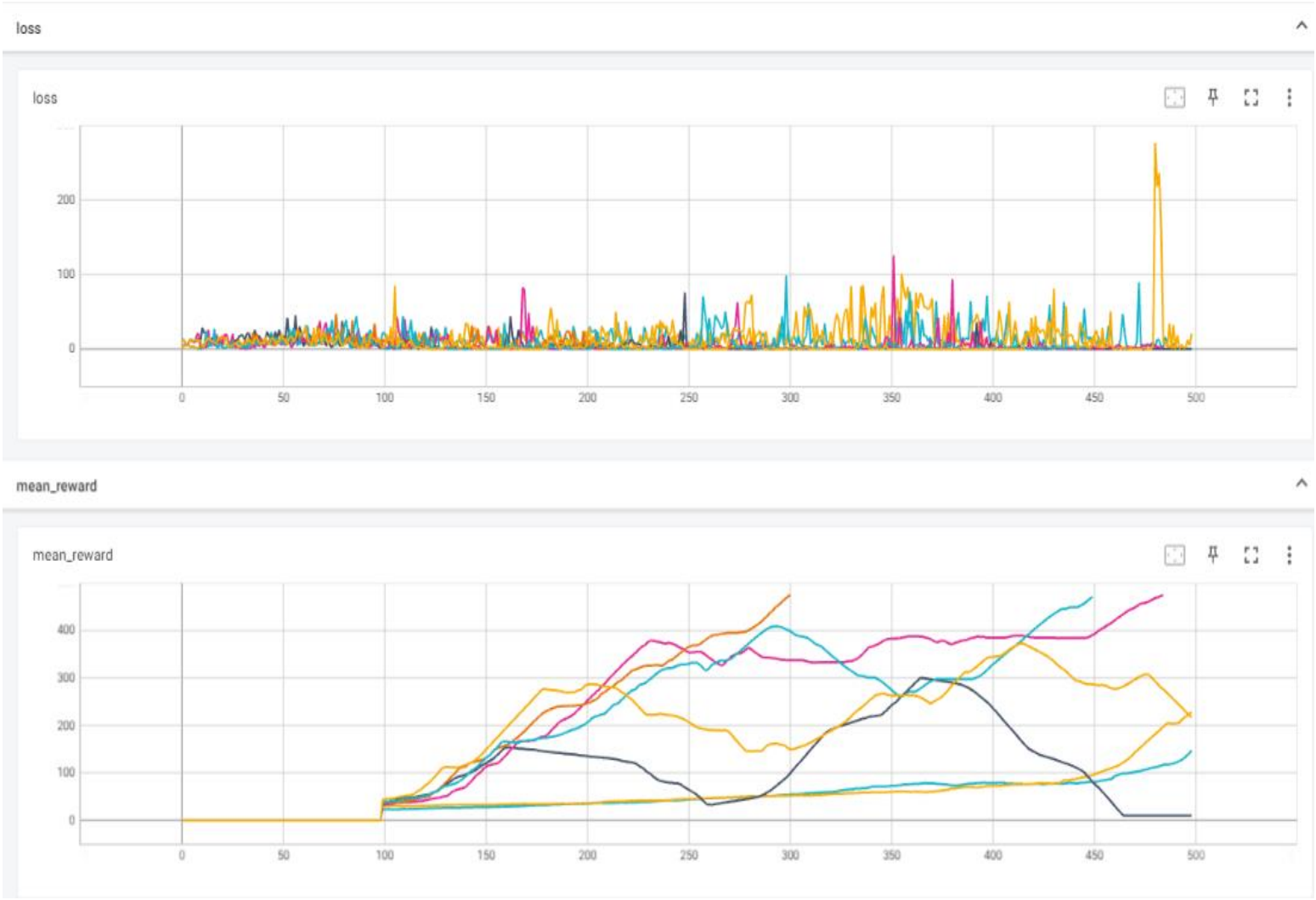
Hyper-parameters Tuning

1. **learning rate of critic-value network tuning**:
   reference values for tuning: actor learning rate: 0.0009, discount factor: 0.99.
   we can notice that for the learning rate of **lr=0.004** the model **successfully** converged into the desired goal after **300** episodes.



Run	Value	Step	Time	Relative
sv_lr=0.0002	475	338	1/11/23, 11:55 PM	1.046 sec
sv_lr=0.004	474.3	300	1/11/23, 11:55 PM	910.1 ms
sv_lr=0.005	473.9	390	1/11/23, 11:55 PM	1.183 sec
sv_lr=0.0001	473.2	438	1/11/23, 11:55 PM	1.37 sec
sv_lr=0.002	450.7	498	1/11/23, 11:55 PM	1.51 sec
sv_lr=0.0008	436	498	1/11/23, 11:55 PM	1.524 sec
sv_lr=0.001	384.5	498	1/11/23, 11:55 PM	1.523 sec
sv_lr=0.003	223.2	498	1/11/23, 11:55 PM	1.493 sec
sv_lr=0.006	196.3	498	1/11/23, 11:55 PM	1.51 sec
sv_lr=0.008	130	498	1/11/23, 11:55 PM	1.528 sec

2. **learning rate of actor-policy tuning**:
   reference values for tuning: critic learning rate: 0.004, discount factor: 0.99.
   for learning rate **lr=0.0009** the agent converged into the desired goal after **300** episodes.



Run	Value	Step	Time	Relative
policy_lr=0.0009	474.3	300	1/11/23, 11:57 PM	933.7 ms
policy_lr=0.0007	474	484	1/11/23, 11:57 PM	1.408 sec
policy_lr=0.001	471.3	449	1/11/23, 11:56 PM	4.279 sec
policy_lr=0.0004	227	498	1/11/23, 11:57 PM	1.399 sec
policy_lr=5e-05	217.3	498	1/11/23, 11:57 PM	1.506 sec
policy_lr=0.0003	146.5	498	1/11/23, 11:57 PM	1.411 sec
policy_lr=0.0006	9.33	498	1/11/23, 11:57 PM	1.413 sec

Optimal hyper-parameters:

Learning rate – policy (actor)	Learning rate – value (critic)	Discount factor
0.0009	0.004	0.99

Time for convergence:

Solved at episode: 300, Algorithm actor_critic for CartPole-v1 converged after 139.630 seconds.

1. The fine-tuning of the model yielded a minor improvement in the model performance (convergence after 313 episodes in comparison to 300 episodes in the tuned model) and almost equivalent convergence time (142 [s] compared to 139.6).
2. We can observe that the tuned model has been optimally converged with different hyperparameters that have been used in the original model while using the original parameters yields a model that not even converging to the desired goal for a maximal number of 500 episodes.
3. We can figure out that the modest improvement in the results is due to the ability of the fine-tuning procedure to leverage the model that has been successfully trained over the Acrobot environment and learned the dynamics of the environment(i.e. the physics of the problem) into the Cartpole environment.

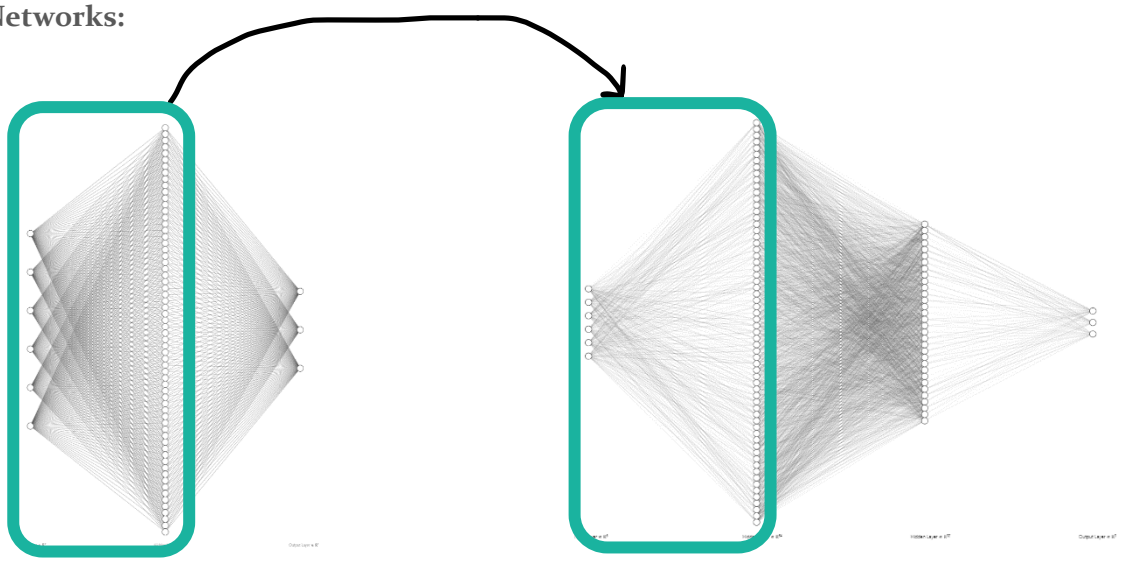
Running the script:
To run the above simulation, run the script **actor_critic.py**, and set the following parameters in the main section:

a. env_name = 'CartPole-v1'
b. policy_learning_rate=0.0009
c. sv_learning_rate = 0.004
d. fine_tuning = True
e. weights_owner_env = 'Acrobot-v1'

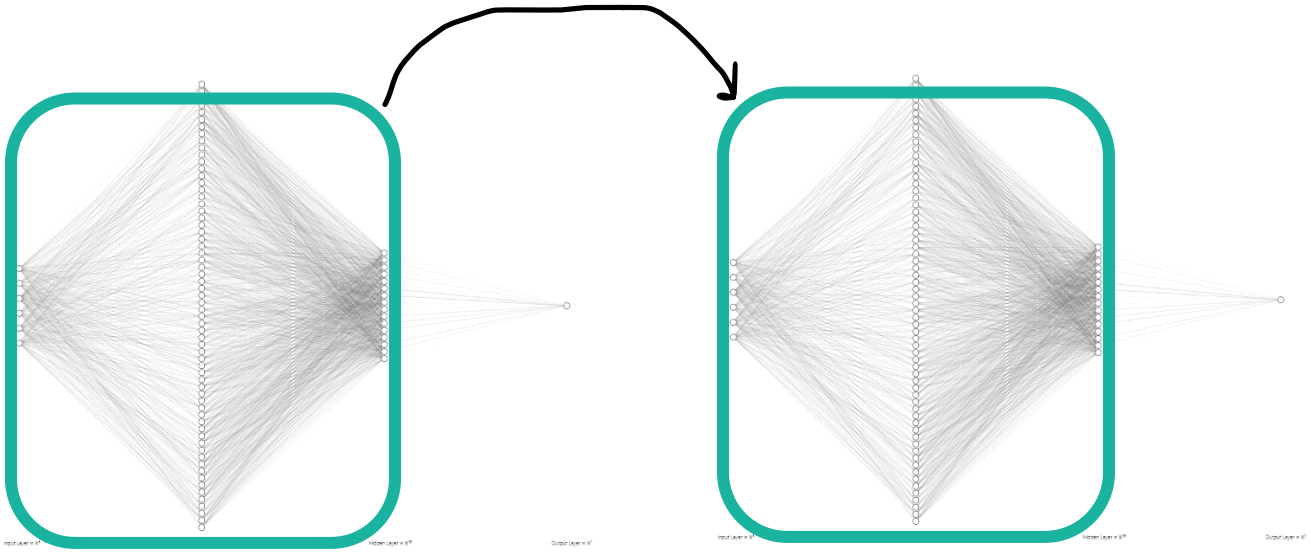
1.5 Fine-tuning of "MountainCarContinuous-v0" environment based on Cartpole -v1 model

In this section, we will observe the architecture that has been trained on the "Cartpole-v1" environment and implant its first hidden layer weights in the architecture that is intended to be trained over the "MountainCarContinuous-v0".

Policy Networks:

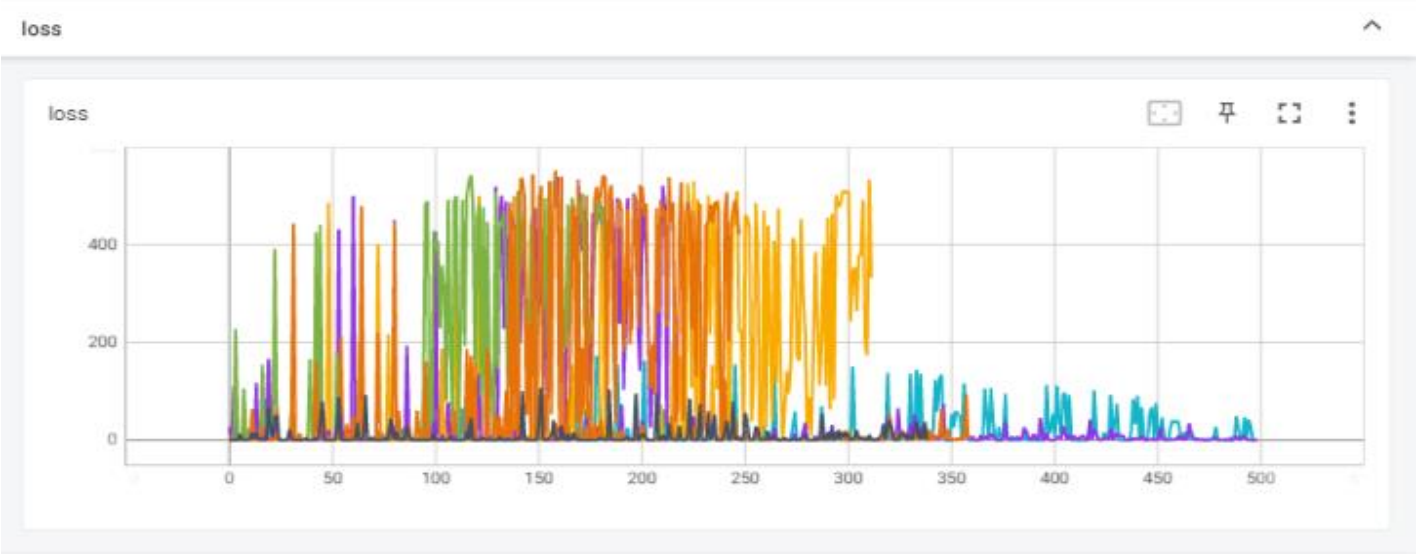


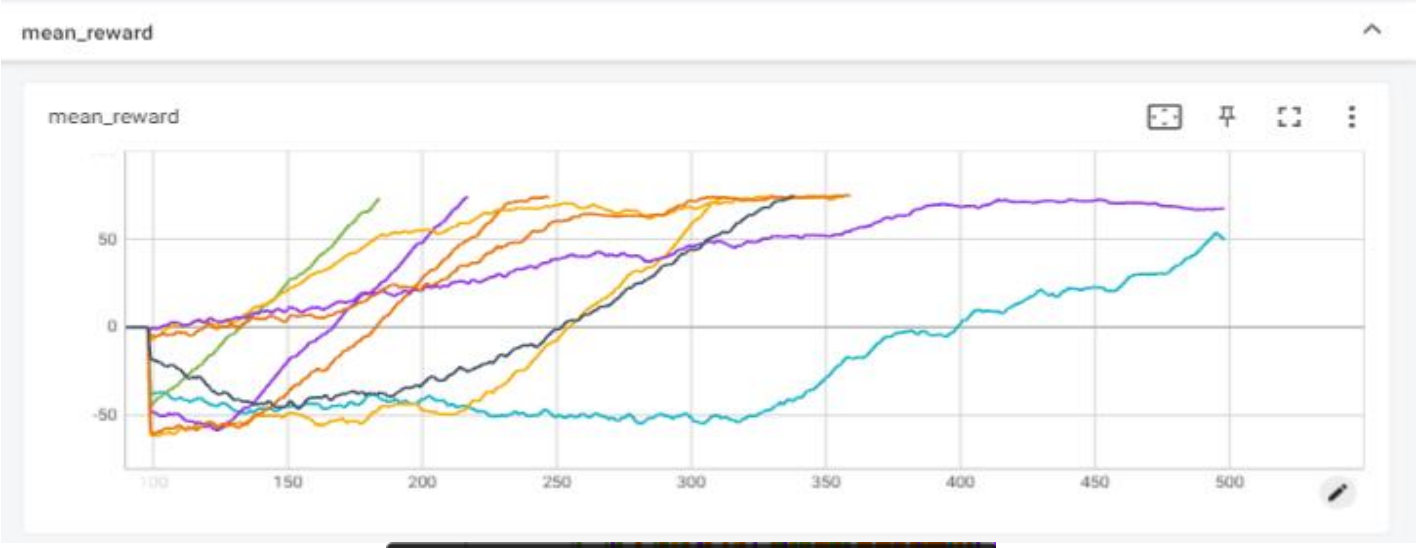
State-Value networks:



Hyper-parameters Tuning

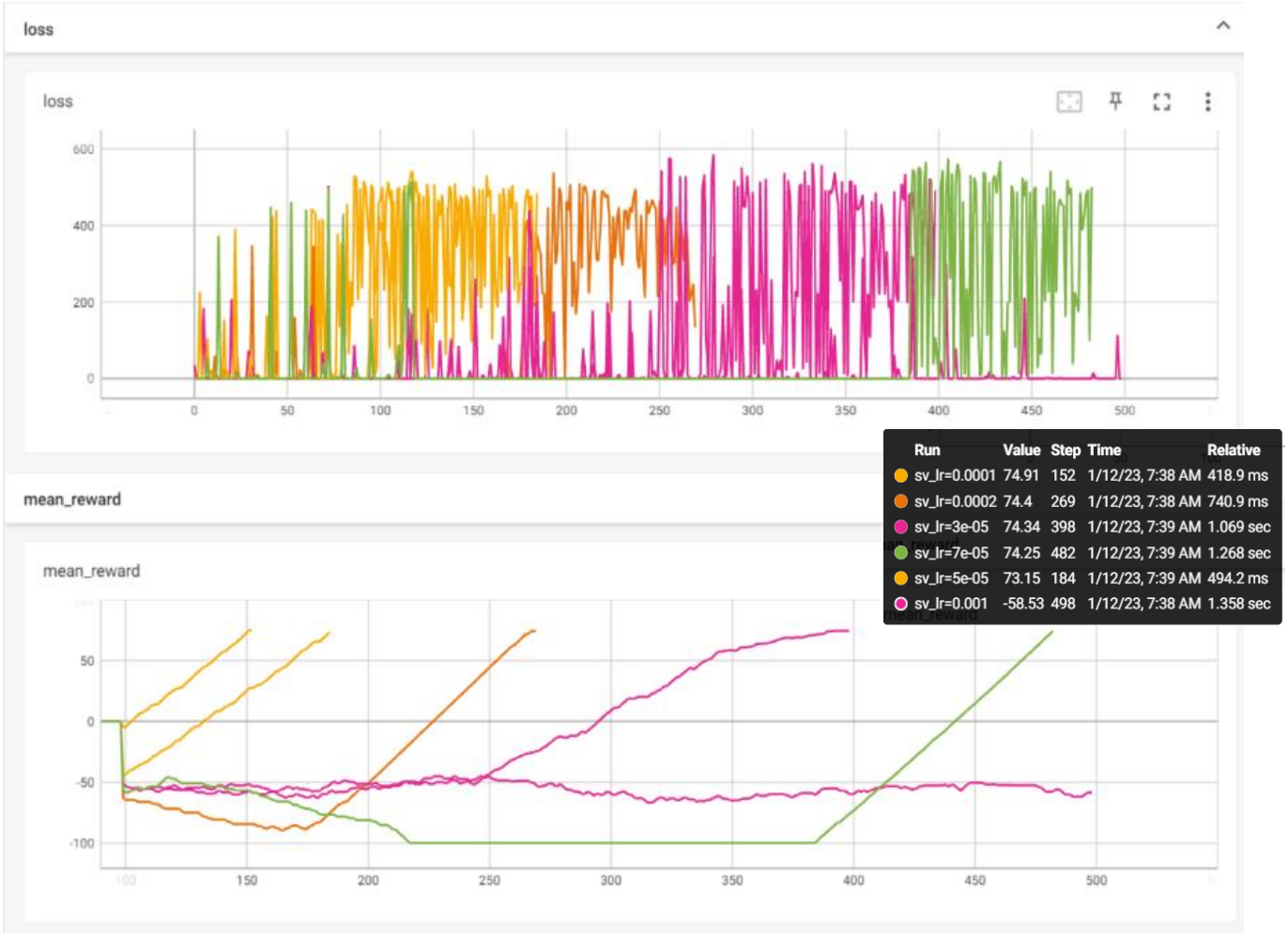
1. **learning rate of actor- policy tuning**:
   reference values for tuning: critic learning rate: 0.0005, discount factor: 0.99.
   for learning rate **lr=0.004** the agent converged into the desired goal after **300** episodes.





Run	Value	Step	Time	Relative
policy_lr=2e-05	75	357	1/11/23, 10:33 PM	940.2 ms
policy_lr=5e-05	74.78	359	1/11/23, 10:33 PM	950.8 ms
policy_lr=7e-05	74.49	338	1/11/23, 10:33 PM	873.2 ms
policy_lr=0.02	73.98	247	1/11/23, 10:33 PM	668 ms
policy_lr=0.002	73.87	311	1/11/23, 10:33 PM	823 ms
policy_lr=0.003	73.7	217	1/11/23, 10:33 PM	573.5 ms
policy_lr=0.004	73.15	184	1/11/23, 10:33 PM	477.1 ms
policy_lr=3e-05	67.19	498	1/11/23, 10:33 PM	1.335 sec
policy_lr=0.0001	49.39	498	1/11/23, 10:33 PM	1.319 sec

2. **learning rate of critic-value network tuning**:
   reference values for tuning: actor learning rate: 0.004, discount factor: 0.99.
   we can notice that for the learning rate of **lr=0.0001** the model is **successfully** converged into the desired goal after **152** episodes.



Optimal hyper-parameters:

Learning rate – policy (actor)	Learning rate – value (critic)	Discount factor
0.004	0.0001	0.99

Time for convergence:

Solved at episode: 152, Algorithm actor_critic for CartPole-v1 converged after 139.630 seconds.

1. The fine-tuned model didn't outperform the individual model that has been trained for the MountainCar environment, reaching the desired goal after 152 episodes (25 more episodes than the original model).
2. In a timing manner, the tuned model converged faster than the individual model (139 vs 151.8 sec), which is straightforward from the tuning process (which his initial weights are already applicable to model some environment dynamics).
3. As before, the tuned model has been optimally converged with different hyperparameters that have been used in the original model.

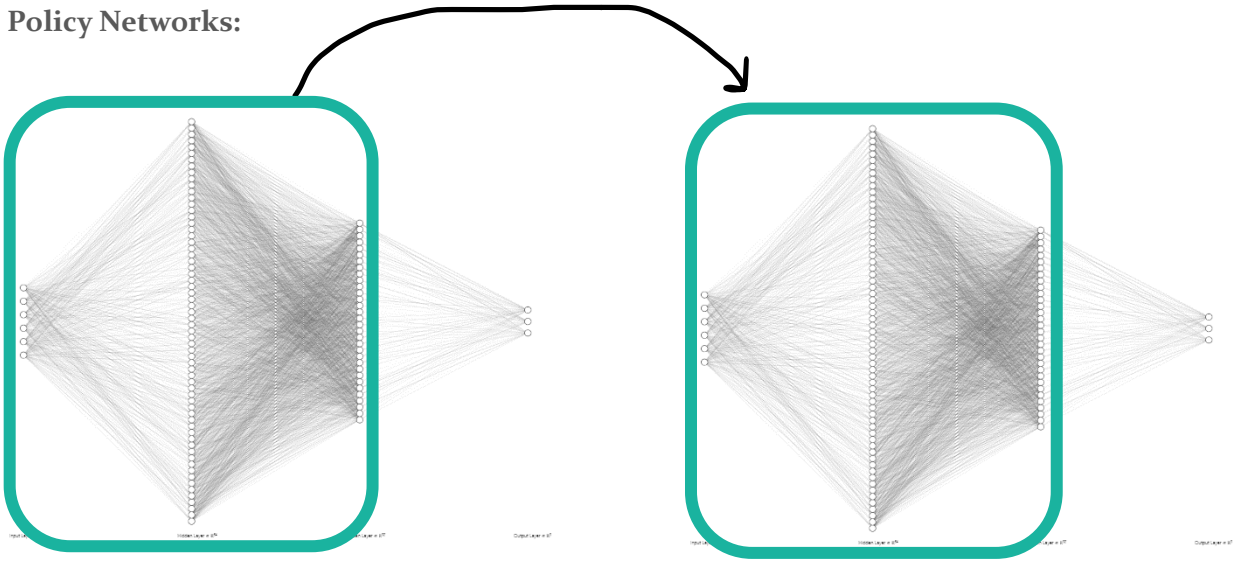
Running the script:
To run the above simulation, run the script **actor_critic_mountain_car.py**, and set the following parameters in the main section:

a. env_name = "MountainCarContinuous-v0"
b. policy_learning_rate=0.004
c. sv_learning_rate = 0.0001
d. fine_tuning = True
e. weights_owner_env = 'CartPole-v1'

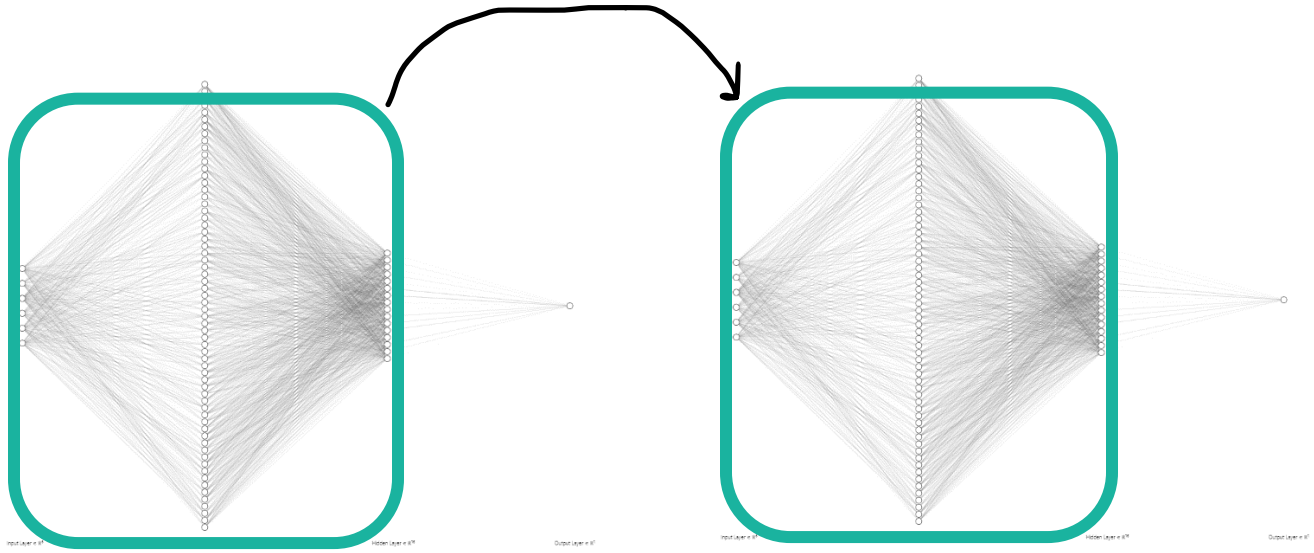
1.6 Fine-tuning of "MountainCarContinuous-v0" environment based on Acrobot-v1 model

Even though the fine-tuning from the cartpole model achieves minor improvement (in the manner of converging time) in the ability of the model to deal with the Mountain car environment, we have managed to reduce the convergence time and achieve better performances (In an episodic manner) over the tested environment by fine-tuning based on the trained Acrobot model. As before we tuned the policy and state-value trained hidden layers into the proposed model that intended to be trained over the "MountainCarContinuous-v0".

Policy Networks:

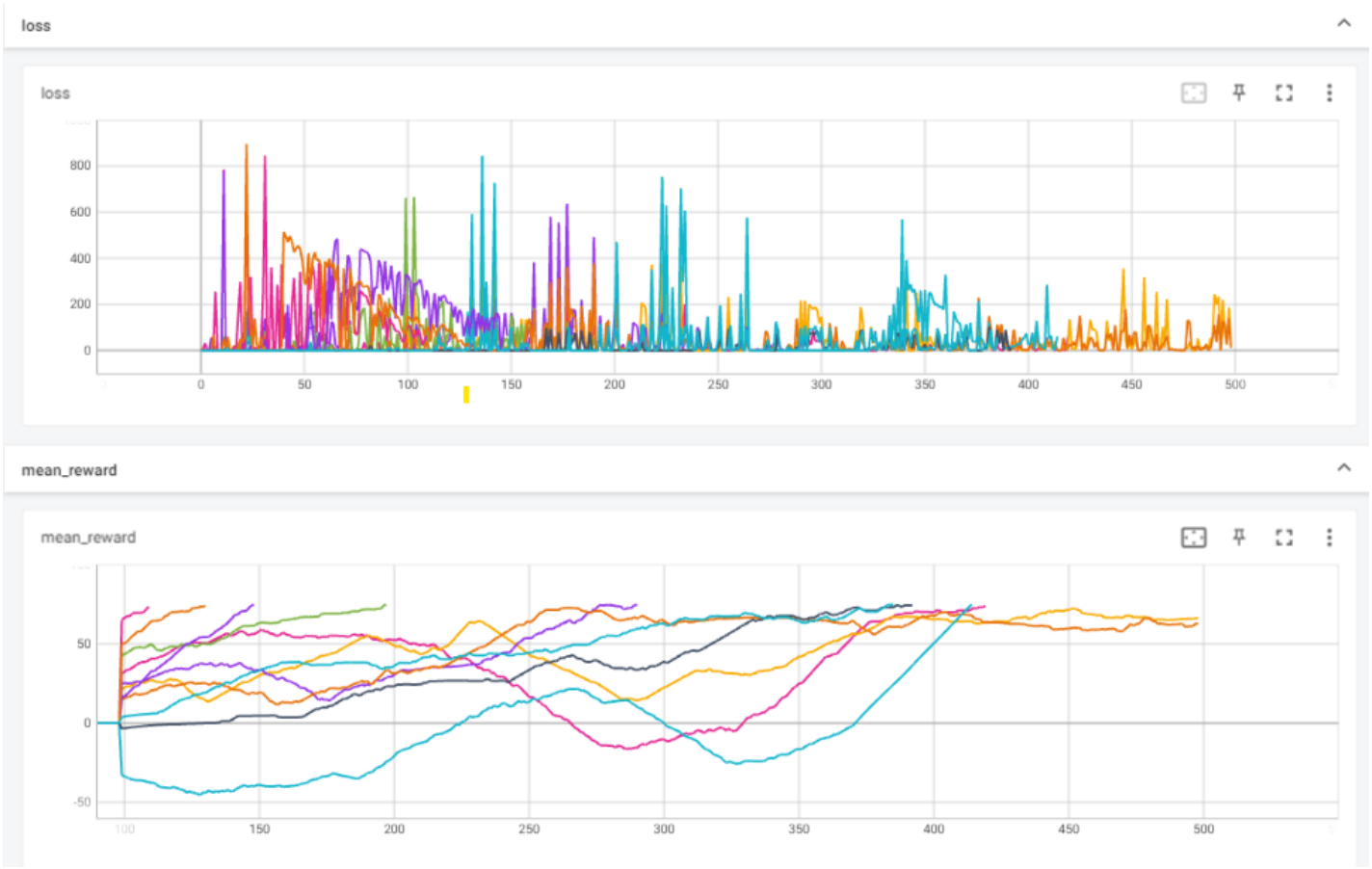


State-Value networks:



Hyper-parameters Tuning

1. **learning rate of actor- policy tuning**:
   reference values for tuning: critic learning rate: 5e-5, discount factor: 0.99.
   we can observe that by only tuning the learning rate of the policy network we already managed to achieve better performances than from the original model - for learning rate **lr=0.002** the agent converged into the desired goal after **109** episodes.



Run	Value	Step	Time	Relative
policy_lr=0.001_Acrobot	74.92	414	1/9/23, 9:18 PM	1.089 sec
policy_lr=0.0004_Acrobot	74.89	197	1/9/23, 9:18 PM	509 ms
policy_lr=7e-05_Acrobot	74.57	385	1/9/23, 9:18 PM	965.8 ms
policy_lr=0.005_Acrobot	74.55	148	1/9/23, 9:18 PM	375.2 ms
policy_lr=0.0003_Acrobot	74.53	290	1/9/23, 9:18 PM	764.7 ms
policy_lr=5e-05_Acrobot	74.31	392	1/9/23, 9:18 PM	1.019 sec
policy_lr=0.0001_Acrobot	74.02	419	1/9/23, 9:18 PM	1.082 sec
policy_lr=0.009_Acrobot	73.77	130	1/9/23, 9:18 PM	332.7 ms
policy_lr=0.002_Acrobot	73.45	109	1/9/23, 9:18 PM	283.5 ms
policy_lr=0.0002_Acrobot	66.24	498	1/9/23, 9:18 PM	1.343 sec
policy_lr=0.0006_Acrobot	62.93	498	1/9/23, 9:18 PM	1.283 sec

Optimal hyper-parameters:

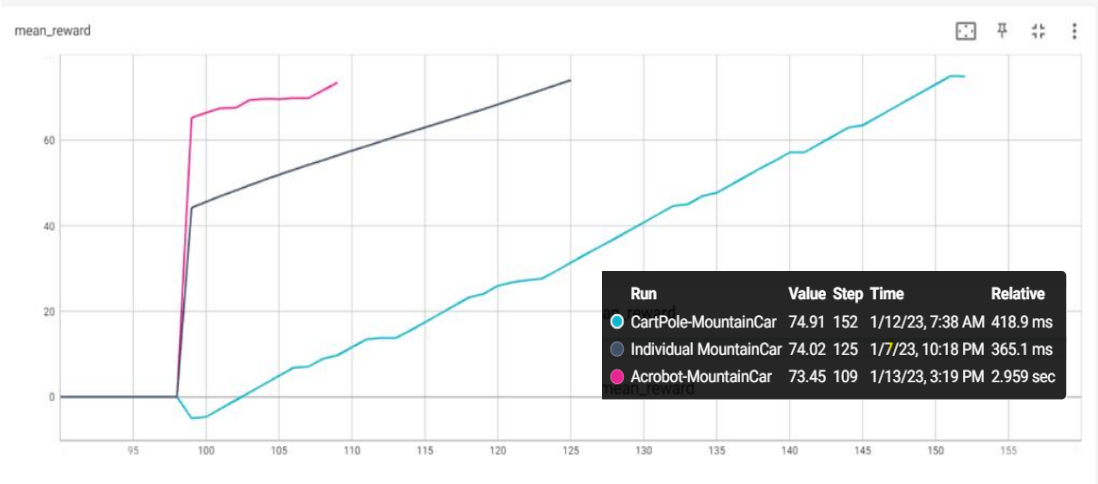
Learning rate – policy (actor)	Learning rate – value (critic)	Discount factor
0.002	5e-5	0.99

Time for convergence:

Solved at episode: 109, Algorithm actor_critic for CartPole-v1 converged after 118.762 seconds.

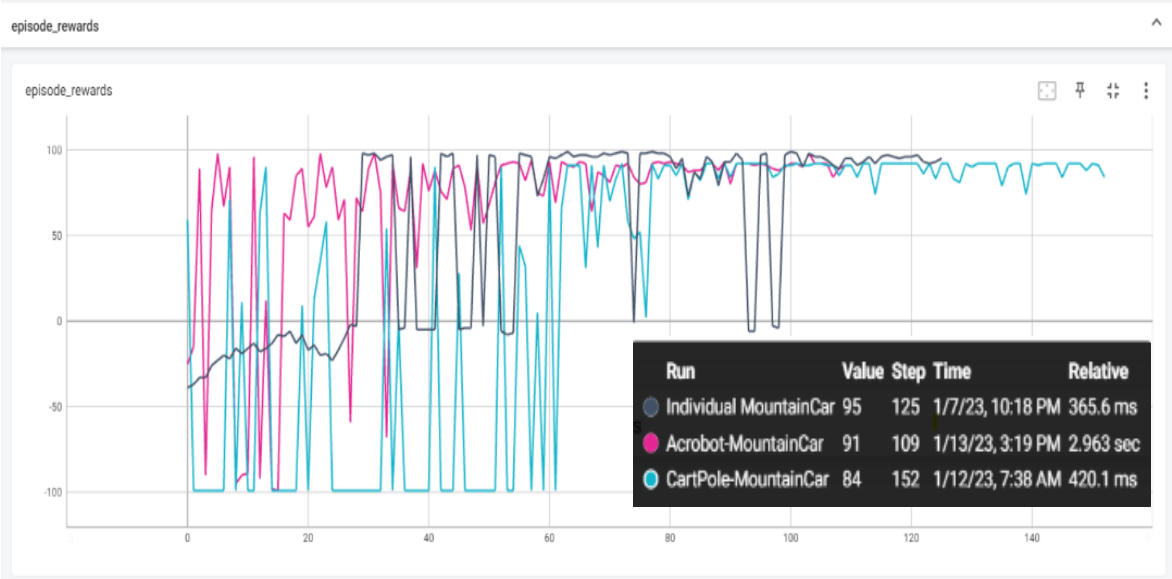
Model comparison

1. In the manner of performances, the acrobat->MountaionCar tuned model has outperformed both the individual and the cartpole->MountaionCar tuned model, achieving the desired goal after only 109 episodes.



2. Moreover, In the manner of convergence time, the acrobat->MountaionCar converged faster to the desired goal, after 118 [sec], ~20 sec less than cartpole->MountaionCar model, and ~ 40 sec less than the individual trained model for MountaionCar.
   On average, each episode converged after ~1.08 seconds, whereas for cartpole->MountaionCar model each episode converged after ~0.918 seconds and for the individual model mean time of a single episode is 1.204 seconds (all the models been trained over GPU: Nvidia RTX 3070 8Gb)
3. This time, optimal results were given for the same policy learning rate as used in the training procedure of the individual model.

4. We can infer from the **episode rewards plot below,** that the better performances achieved by the acrobat->MountaionCar tuning procedure are probably due to the high rewards in the initial episodes, which are induced by the weights of the trained acrobat model. The high rewards at the beginning plus the fine-tuning procedure also led to faster and more stabilized rewards in the following episodes.



Running the script:
To run the above simulation, run the script **actor_critic_mountain_car.py**, and set the following parameters in the main section:

a. env_name = "MountainCarContinuous-v0"
b. policy_learning_rate=0.002
c. sv_learning_rate = 5e-5
d. fine_tuning = True
e. weights_owner_env = 'Acrobot-v1'

Section 3 – Transfer Learning

In this section, we will implement the progressive network method over a specific environment, where the architecture allows us to induce the trained model capabilities into the current one, hopefully achieving a better understanding of the dynamics of the problem and yielding faster convergence.

As we learned in class, the connections between the pre-trained models' hidden layers, which resemble previously extracted and learned features of each environment, are intended to immune the model to forgetting and leverage prior knowledge

2.1 Progressive network over the "MountainCarContinuous-v0" environment

in this section, we will use the weights and layers of the pre-trained models for the "Cartpole" and the "Acrobot" environments, freeze them and connect their hidden layers to the trained one to generate the progressive network architecture for the "MountainCarContinuous" Environment.

Proposed Architecture - Policy Network(Actor):

We used the policy network architectures of the individual trained models, as we presented in question 1.

1. "**Acrobot**" architecture – **Policy Network**:

   Input: dense layer, **size**: 6

   Hidden 1 (h1_ac): dense layer, **size**: 64

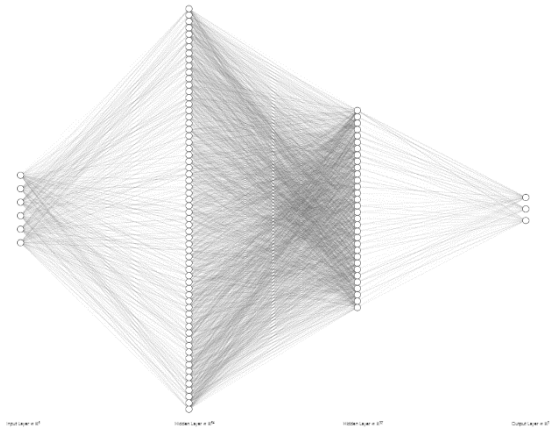
   Activation: ReLU

   Hidden 2 (h2_ac): dense layer, **size**: 32

   Activation: ReLU

   Output: dense layer, **size**: 3

   Activation: Linear



2. "CartPole" architecture – **Policy Network**:

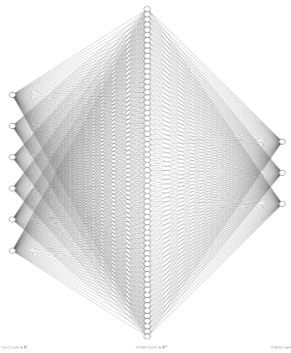
   Input: dense layer, **size**: 6

   Hidden 1 (h1_cp): dense layer, **size**: 64

   Activation: ReLU

   Output: dense layer, **size**: 3

   Activation: Linear



3. "**MountainCar**" architecture – **Policy Network**:

   Input: dense layer, **size**: 6

   Hidden 1 (h1_ac): dense layer, **size**: 64

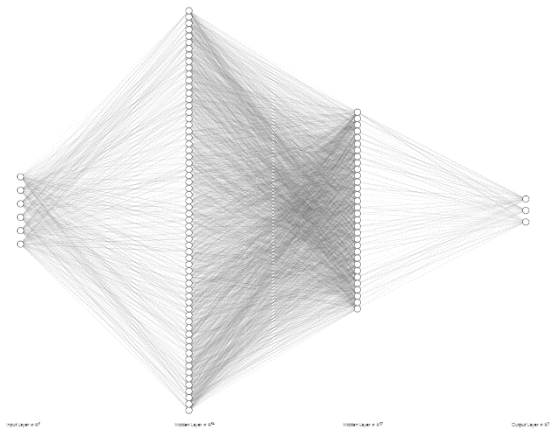
   Activation: ReLU

   Hidden 2 (h2_ac): dense layer, **size**: 32

   Activation: ReLU

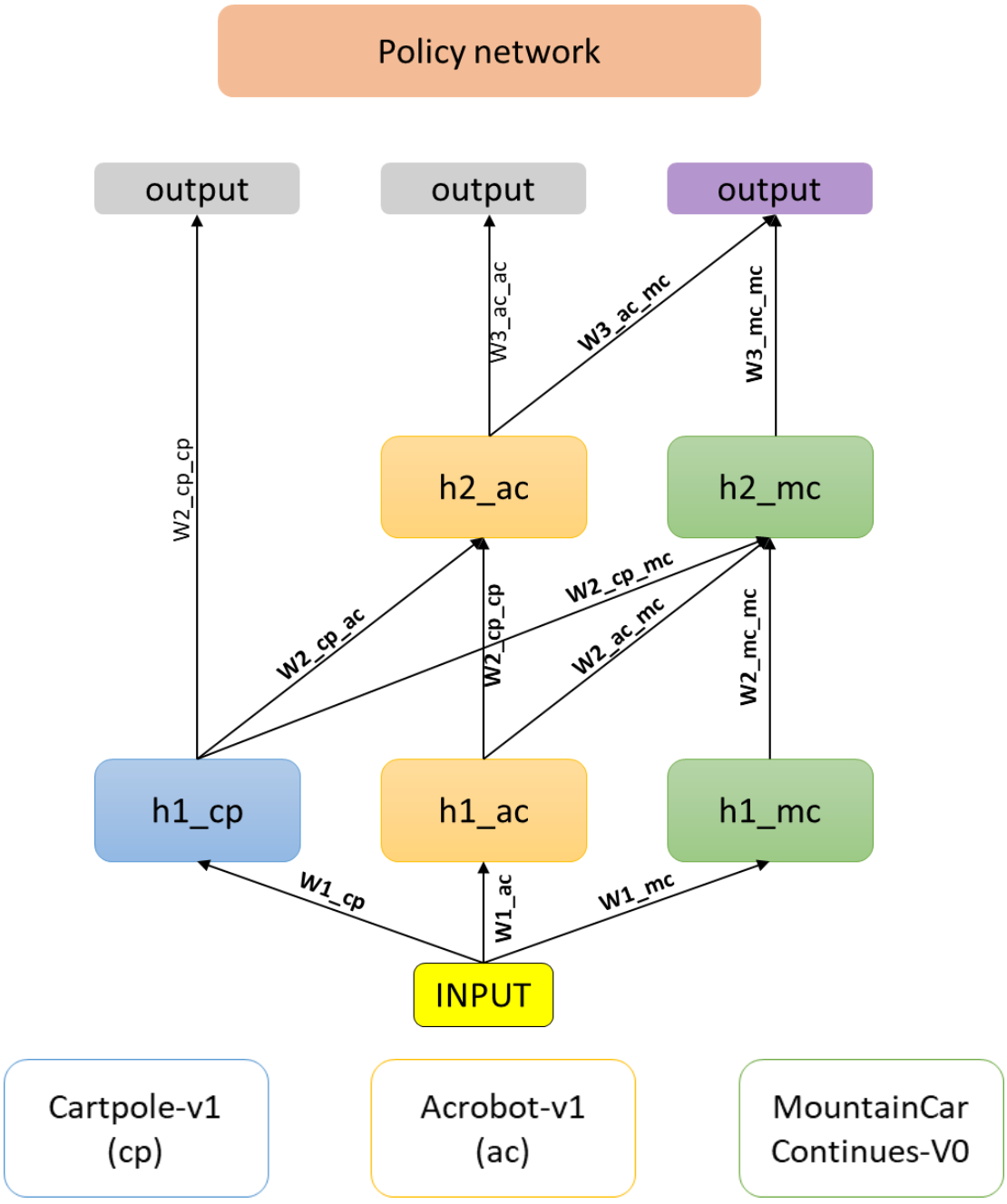
   Output: dense layer, **size**: 3

   Activation: Linear



4. **Progressive network architecture:**
   The hidden layers of the trained models above are frozen and used as building blocks for the progressive network. The connections between the layers and the "mountain

Car" model are the new weights in the architecture. As present in the following scheme:



We have connected the single hidden layer of the "Cartpole" trained model into the second hidden layer of the "Acrobat", for gathering their hidden layer features together, in order to hopefully create a better representation of the dynamics (mixed from both environments).

Only the bold weights (each line connects between blocks in the architecture) are trained.

Proposed Architecture – State-value network(Critic):

As we have already presented, all the environments have been trained over the same architecture:

1. **State-Network**:

   Input: dense layer, **size**: 6

   Hidden 1 (h1): dense layer, **size**: 64

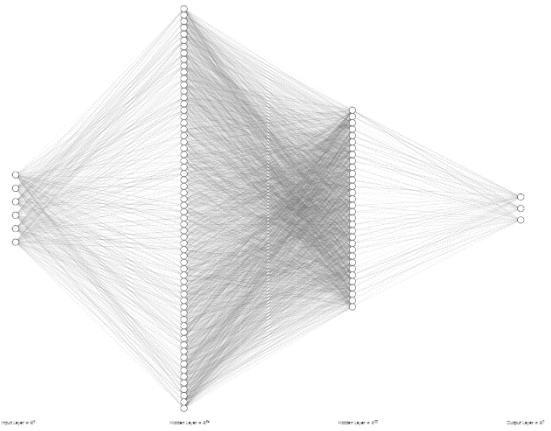
   Activation: ReLU

   Hidden 2 (h2): dense layer, **size**: 16

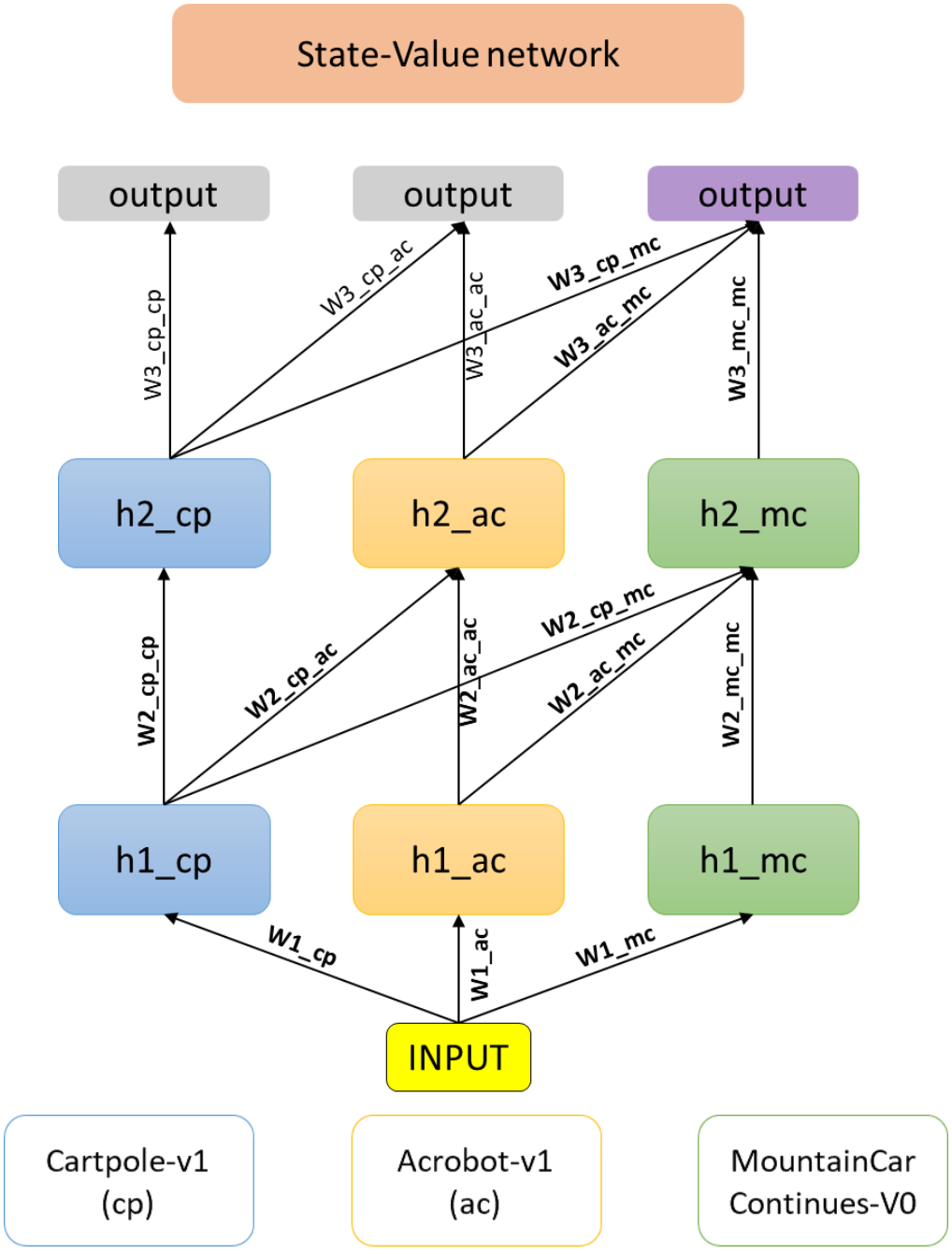
   Activation: ReLU

   Output: dense layer, **size**: 1

   Activation: Linear



2. **Progressive network architecture:**
   Only the **bold weights** (each line connect between blocks in the architecture) are trained.



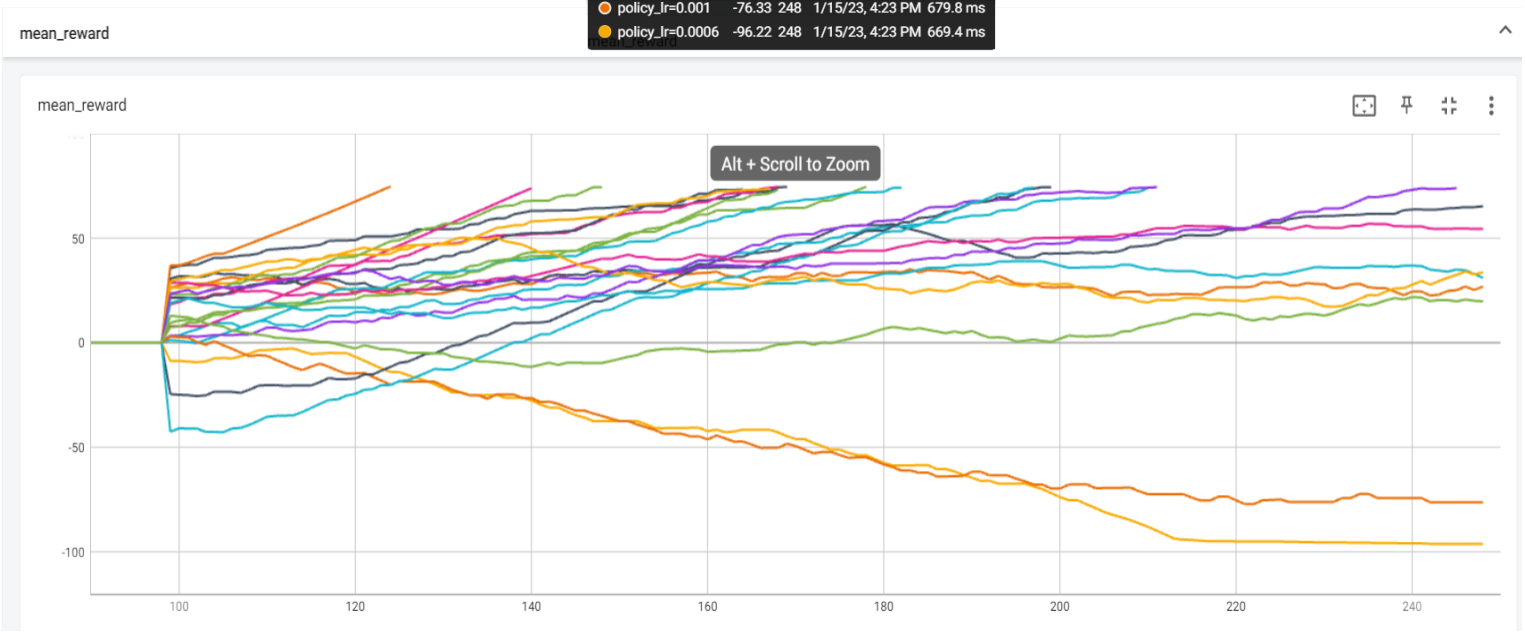
Only the **bold weights** (each line connects between blocks in the architecture) are the ones trained.

Hyper-parameters Tuning

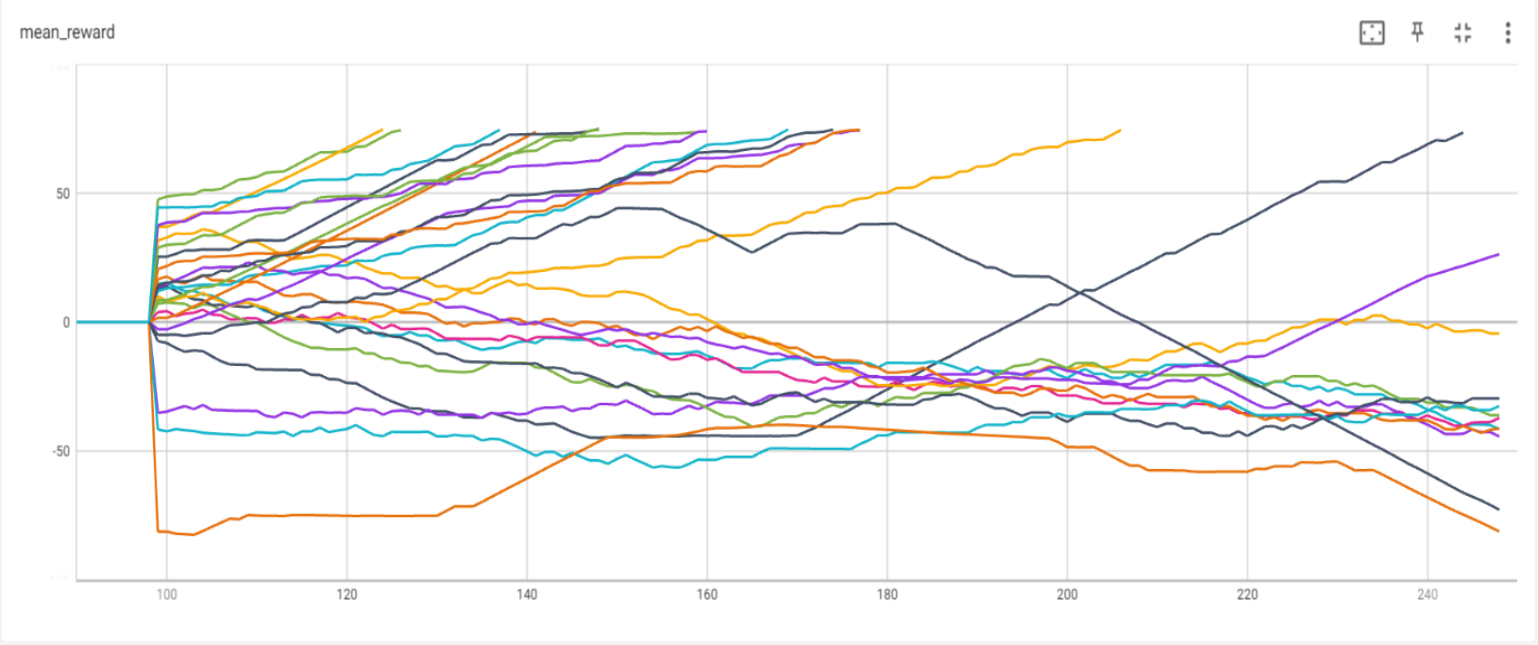
The actor-critic algorithm is implemented over the discussed environment to reach a mean reward of 75 points for 100 consecutive episodes.
We set the same seed for optimal tuning through all the tuning procedures.
(maximal number of episodes in each simulation is limited to 250 now)

1. **learning rate of actor-policy tuning**:
reference values for tuning: critic learning rate: 0.005, discount factor: 0.99.
for learning rate **lr=0.00045** the agent converged into the desired goal after **125** episodes, and after 97.582 seconds.

Run	Value	Step	Time	Relative
policy_lr=0.00045	74.69	124	1/15/23, 4:27 PM	361.2 ms
policy_lr=0.00047	74.6	178	1/15/23, 4:24 PM	475.7 ms
policy_lr=0.00043	74.54	168	1/15/23, 4:24 PM	464.5 ms
policy_lr=0.00041	74.53	169	1/15/23, 4:24 PM	468.8 ms
policy_lr=0.0007	74.39	211	1/15/23, 4:23 PM	563.7 ms
policy_lr=0.00055	74.37	148	1/15/23, 4:24 PM	421.1 ms
policy_lr=0.002	74.36	199	1/15/23, 4:23 PM	580.2 ms
policy_lr=0.003	74.14	197	1/15/23, 4:23 PM	559.2 ms
policy_lr=0.00051	74.08	182	1/15/23, 4:24 PM	512.9 ms
policy_lr=0.00042	74.06	210	1/15/23, 4:24 PM	573.8 ms
policy_lr=0.00052	73.89	140	1/15/23, 4:24 PM	387.4 ms
policy_lr=7e-05	73.89	245	1/15/23, 4:23 PM	682.7 ms
policy_lr=0.00053	73.61	167	1/15/23, 4:24 PM	460.7 ms
policy_lr=0.00049	73.31	164	1/15/23, 4:24 PM	472 ms
policy_lr=0.008	73.24	168	1/15/23, 4:24 PM	449.4 ms
policy_lr=0.0002	65.24	248	1/15/23, 4:23 PM	678.5 ms
policy_lr=4e-05	54.42	248	1/15/23, 4:23 PM	684.9 ms
policy_lr=6e-05	33.64	248	1/15/23, 4:23 PM	675.9 ms
policy_lr=5e-05	31.03	248	1/15/23, 4:23 PM	704.2 ms
policy_lr=0.0001	26.9	248	1/15/23, 4:23 PM	686.9 ms
policy_lr=9e-05	19.78	248	1/15/23, 4:23 PM	691.5 ms
policy_lr=0.001	-76.33	248	1/15/23, 4:23 PM	679.8 ms
policy_lr=0.0006	-96.22	248	1/15/23, 4:23 PM	669.4 ms



2. learning rate of policy network tuning:
reference values for tuning: actor learning rate: 0.00045, discount factor: 0.99.
for learning rate **lr=0.005** the agent converged into the desired goal after **125** episodes.



Run	Value	Step	Time	Relative
sv_lr=0.0047	74.83	148	1/15/23, 8:06 PM	429.3 ms
sv_lr=0.005	74.69	124	1/15/23, 8:06 PM	358.7 ms
sv_lr=0.003	74.68	169	1/15/23, 8:06 PM	472.2 ms
sv_lr=0.01	74.59	137	1/15/23, 8:06 PM	397 ms
sv_lr=0.0054	74.58	174	1/15/23, 8:06 PM	490.9 ms
sv_lr=0.006	74.49	148	1/15/23, 8:06 PM	411.7 ms
sv_lr=0.0045	74.48	206	1/15/23, 8:06 PM	592.4 ms
sv_lr=0.007	74.47	177	1/15/23, 8:06 PM	512.9 ms
sv_lr=0.0044	74.36	126	1/15/23, 8:06 PM	369.6 ms
sv_lr=0.0051	74.23	177	1/15/23, 8:06 PM	510.3 ms
sv_lr=0.0055	73.94	160	1/15/23, 8:06 PM	453.5 ms
sv_lr=0.0046	73.77	141	1/15/23, 8:06 PM	404.3 ms
sv_lr=0.0052	73.59	159	1/15/23, 8:06 PM	449.4 ms
sv_lr=0.0004	73.47	244	1/15/23, 8:05 PM	643.9 ms
sv_lr=0.0043	26.26	248	1/15/23, 8:06 PM	737.8 ms
sv_lr=0.0006	-4.461	248	1/15/23, 8:05 PM	755 ms
sv_lr=0.002	-29.56	248	1/15/23, 8:05 PM	677 ms
sv_lr=0.0048	-32.58	248	1/15/23, 8:06 PM	715.1 ms
sv_lr=0.0009	-36.11	248	1/15/23, 8:05 PM	713.6 ms
sv_lr=0.0005	-37.2	248	1/15/23, 8:05 PM	733.6 ms
sv_lr=0.001	-41.26	248	1/15/23, 8:05 PM	703.5 ms
sv_lr=0.00045	-41.5	248	1/15/23, 8:05 PM	668.8 ms
sv_lr=0.0007	-44.37	248	1/15/23, 8:05 PM	706.2 ms
sv_lr=0.008	-72.74	248	1/15/23, 8:06 PM	714 ms
sv_lr=0.0053	-81.19	248	1/15/23, 8:06 PM	726.6 ms

Optimal hyper-parameters:

Learning rate – policy (actor)	Learning rate – value (critic)	Discount factor
0.00045	0.005	0.99

Time for convergence:

Solved at episode: **125**, Algorithm actor_critic for Acrobot-v1 converged after **97.582** seconds. We can notice that the model achieves similar results as the individual MountainCar model in the manner of performances (achieves the desired goal after 126 episodes), but converge **way much faster** than all the trained algorithms for the mountain car environment:

Model	Time passed until for reaching the goal [sec]	Amount of episodes until convergence	Mean time for a single episode [sec]
Progressive network	**97.582**	125	**0.774**
Acrobot->MountainCar	118.762	**109**	1.08
Cartpole->MountaionCar	139.630	152	0.918
Individual MountaionCar	151.823	126	1.204

All the models have been trained over GPU: Nvidia RTX 3070 8Gb

We can infer that the significant improvement in the convergence time is due to the **combined latent representation** of the weighted extracted feature of both trained models, which enables us to represent the dynamics of both the Cart and the Acrobot physics, which intuitively describes the mountain car environment.

It can be also seen by the mean award plot that the average award curve of the progressive model is having the highest slope (as a function of the episode taken) in comparison to other models.
Moreover, observing the rewards per episode, while the progressive model receives lower rewards at the initial episodes(~30 first episodes), it dramatically increases in the following steps, maintaining high reward (above 85) until convergence, without any fluctuations compared to the other models.

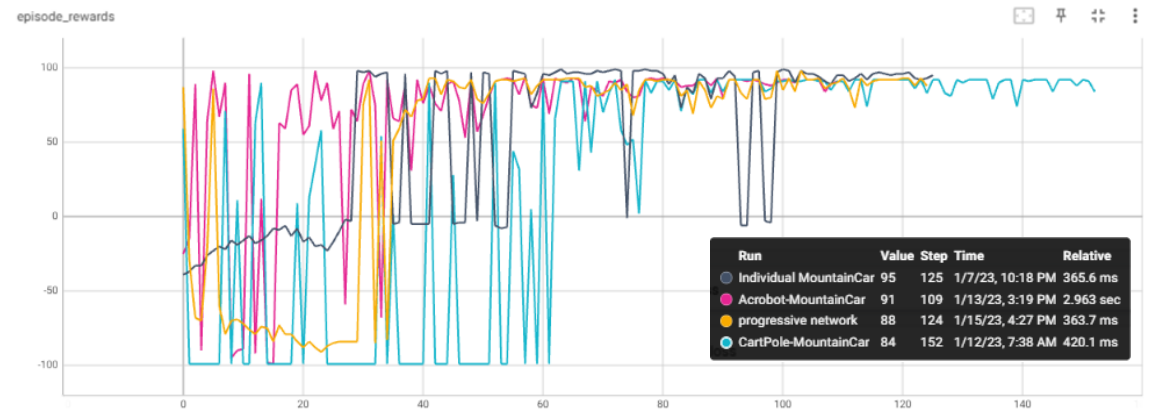


Figure 3 - MountainCar models reward as a function of episode number.

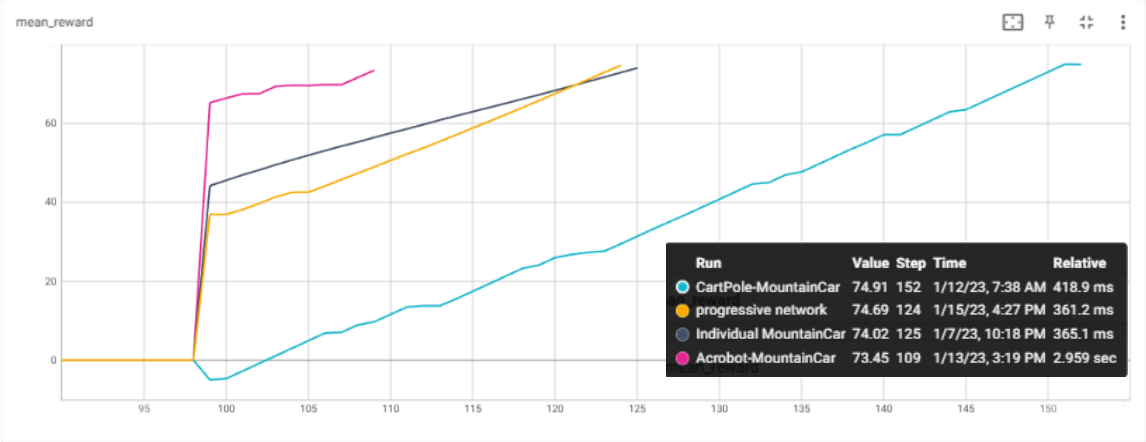


Figure 3 - MountainCar models mean reward as a function of episode number.

Running the script:
To run the above simulation, run the script **progressive_network.py**, and set the following parameters in the main section:

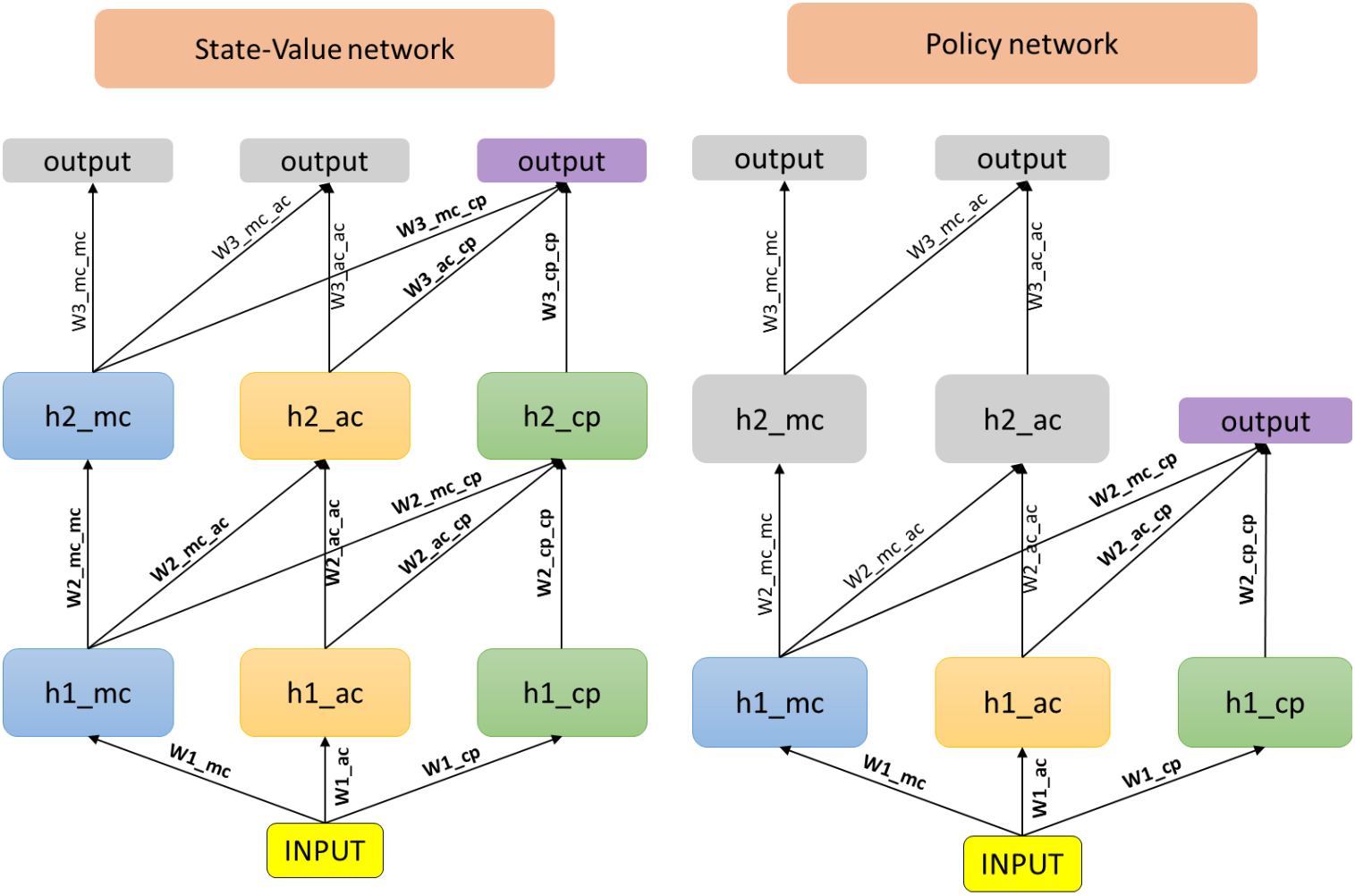
a. env_name = "MountainCarContinuous-vo”

2.2 Progressive network over "Cartpole-v1" environment

in this section, we will use the weights and layers of the pre-trained models for "MountainCarContinuous" and "Acrobot" environments, to generate the progressive network architecture for the "Cartpole" Environment.

Progressive network architecture:

We used the same policy and the state value networks as presented above where again, the hidden layers of the trained models are frozen and used as building blocks for the progressive network. The connections between the layers and the "Cartpole" model are the new weights in the architecture. As present in the following scheme:

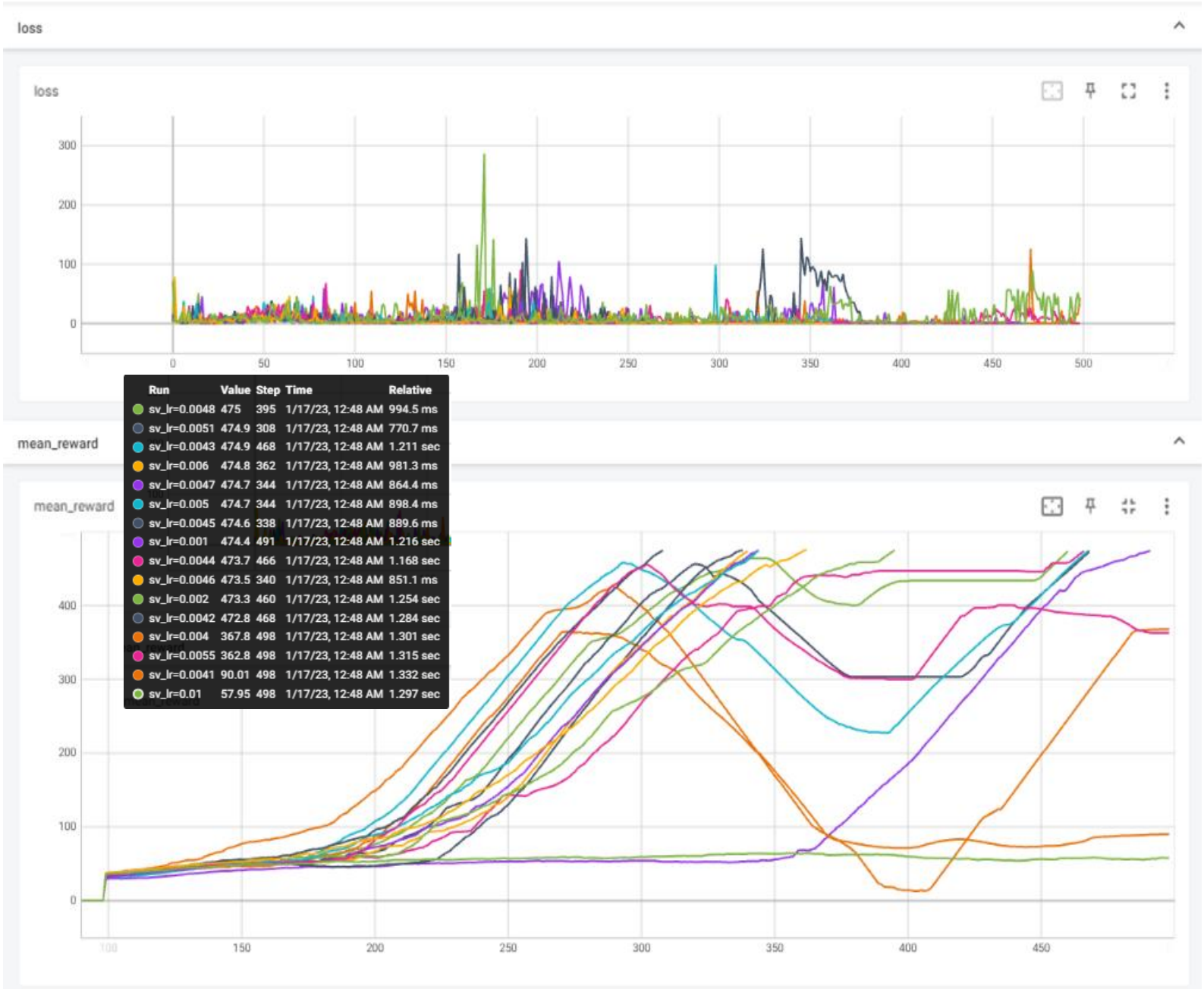


Only the **bold weights** (each line connect between blocks in the architecture) are trained.

Hyper-parameters Tuning

The actor-critic algorithm is implemented over the discussed environment to reach a mean reward of 75 points for 100 consecutive episodes.
We set the same seed for optimal tuning through all the tuning procedures.

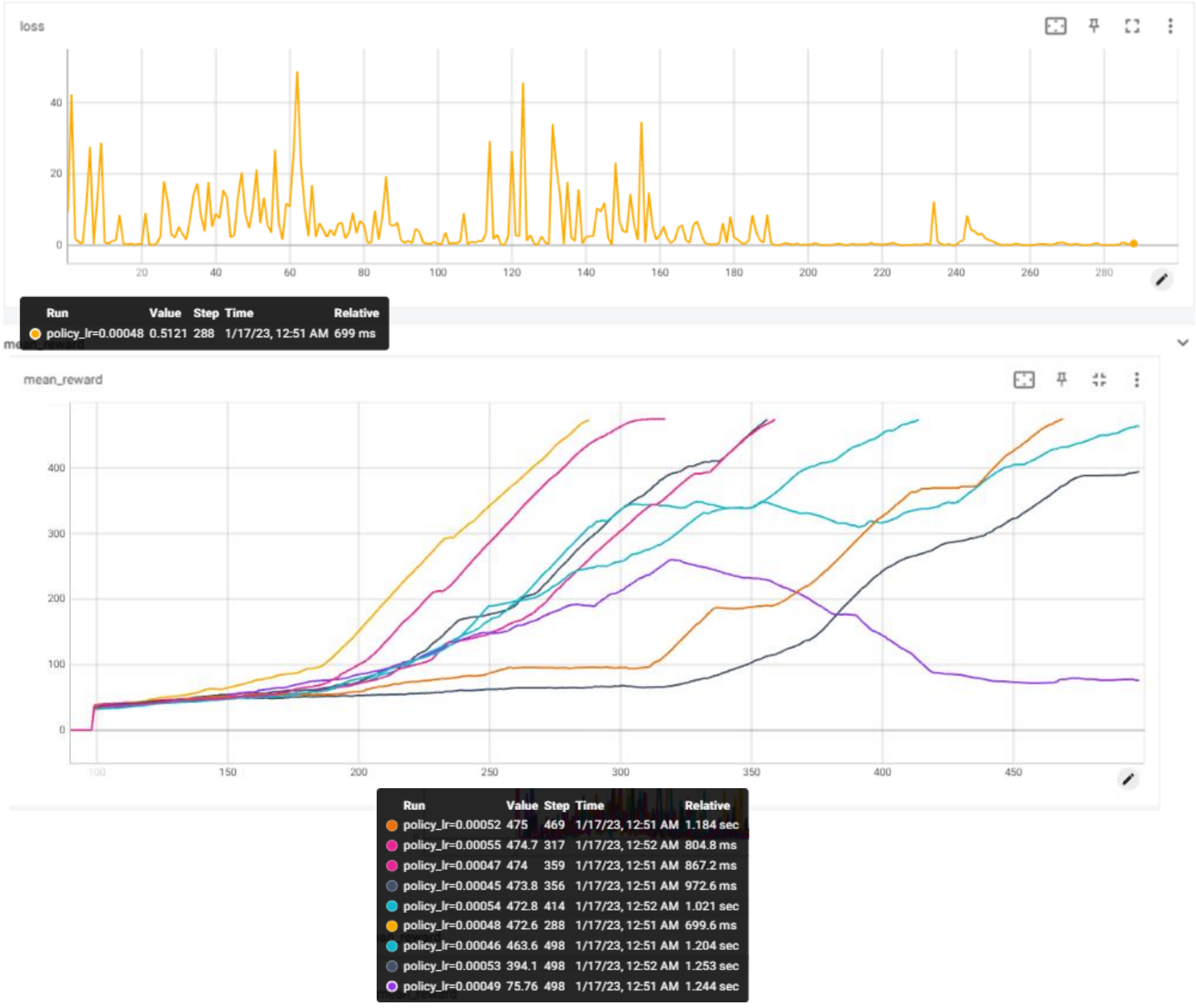
1. **learning rate of actor- policy tuning**:
reference values for tuning: critic learning rate: 0.0005, discount factor: 0.99.
for learning rate **lr=0.0051** the agent converged into the desired goal after **308** episodes.



2. **learning rate of policy network tuning**:
reference values for tuning: actor learning rate: 0.00045, discount factor: 0.99.

Since the state value network has been trained for a given policy learning rate of 0.00005, we will tune around this learning rate for optimal (locally, global not guaranteed) results.

for learning rate **lr=0.00048** the agent converged into the desired goal after **288** episodes.



Optimal hyper-parameters:

Learning rate – policy (actor)	Learning rate – value (critic)	Discount factor
0.00048	0.0051	0.99

Time for convergence:

Solved at episode: 289, Algorithm actor_critic for CartPole-v1 converged after 95.466 seconds.

First, in the manner of performances, the progressive network for the "cartpole" environment has converged after 289 episodes, which are ~10 episodes **less** than fine-tuned model and ~20 episodes less than the individual model.

As before, The model has converged **way much faster** than all the trained algorithms for the Cartpole environment:

Model	Time passed until for reaching the goal [sec]	Amount of episodes until convergence	Mean time for a single episode [sec]
Progressive network	**95.466**	**289**	**0.33**
Acrobot-> Cartpole	139.630	300	0.4654
Individual Cartpole	142.601	314	0.454

All the models been trained over GPU : Nvidia RTX 3070 8Gb

The improvement in the convergence time is due to the **combined contribution of both the Acrobot and the MountainCar** to the understanding of the Cart dynamics.

As before, from the mean award plot, we can observe that the average award curve of the progressive model is having the highest slope (as a function of the episode taken) in comparison to other models, corresponding to the faster convergence time.
By observing the rewards per episode, after ~175 episodes the progressive network presents sharp rise in the episode, maintaining maximal reward (500) until convergence, with few fluctuations comparing to the other models.

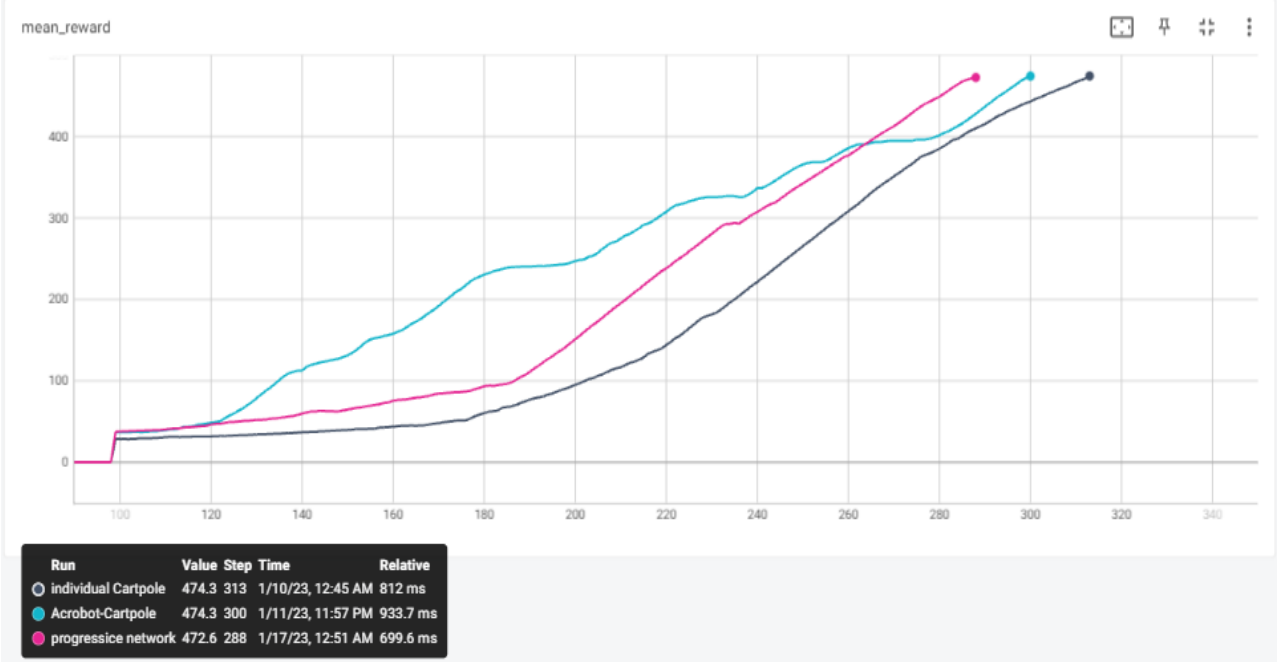


Figure 5 – Cartpole models reward as a function of episode number.

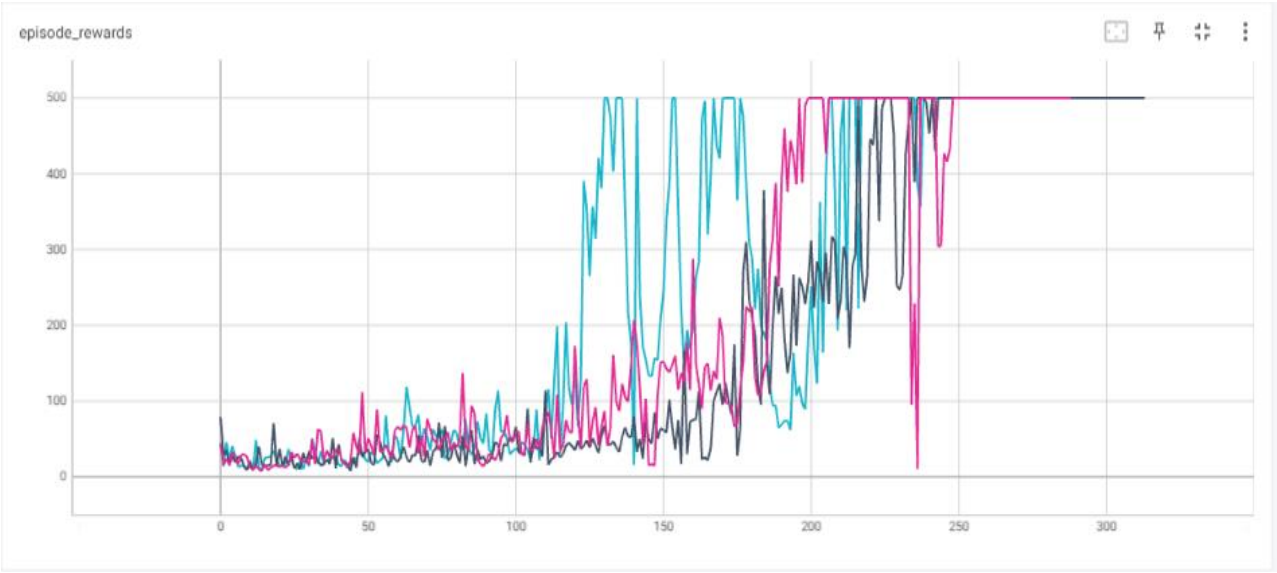


Figure 6 - Cartpole models mean reward as a function of episode number.

Running the script:
To run the above simulation, run the script **progressive_network.py**, and set the following parameters in the main section:

b. env_name = "Cartpole-v1"