Section 1 – Expert Agents

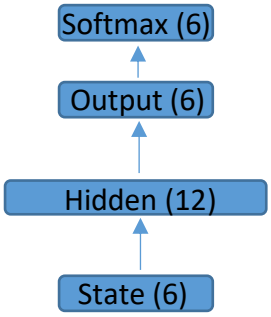
In this section I trained 3 expert Actor-Critic agents, one for each individual task. For the following sections, I used the same size of input, output and number of activations in the hidden layers across all tasks:

- State size (Input): 6
- Actions size (Output): 6
- Hidden layers size: 12

Instead of padding the actions with 0's which could result with several problems, including invalid action selection which would throw the program, or otherwise result in a custom negative reward, I defined a dictionary of valid actions for each task. Each action (of the agent) is used as the key of the dictionary, and its value is one of the actual environment valid actions. Note that I defined the dictionary so under random agent each environment action would have equal probability to be selected.

1. **Cartpole Agent –** *cartpole_expert.py*

   Actor's NN architecture:



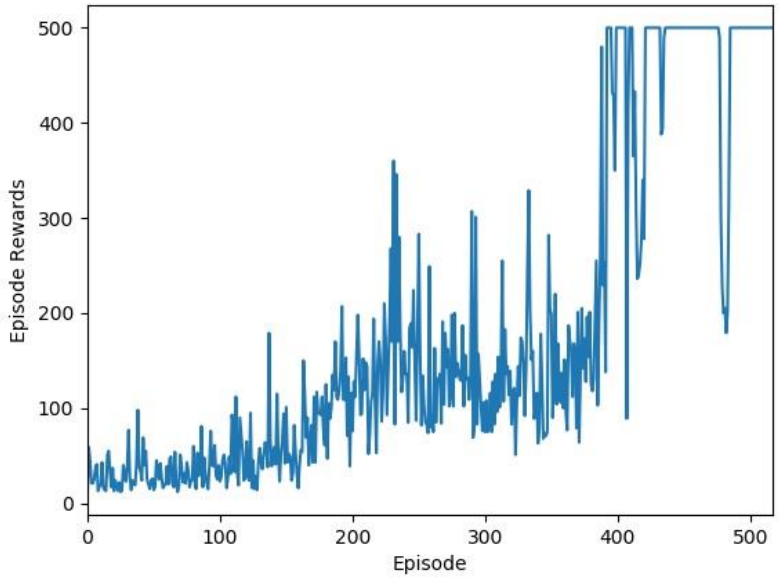
   Settings:
   - **Actions probability distribution**: Softmax
   - **Loss function**: Softmax Cross-entropy
   - **Activation Function**: eLU
   - **Optimizer**: Adam
   - **Number of original actions**: 2 (0, 1)
   - **Actions Dictionary**: {**0**: 0, **1**: 0, **2**: 0, **3**: 1, **4**: 1, **5**: 1}
   - **Critic Network**: Same architecture but with output size of 1 and MSE loss
   - **Solved definition**: Average reward of 475 for 100 consecutive episodes

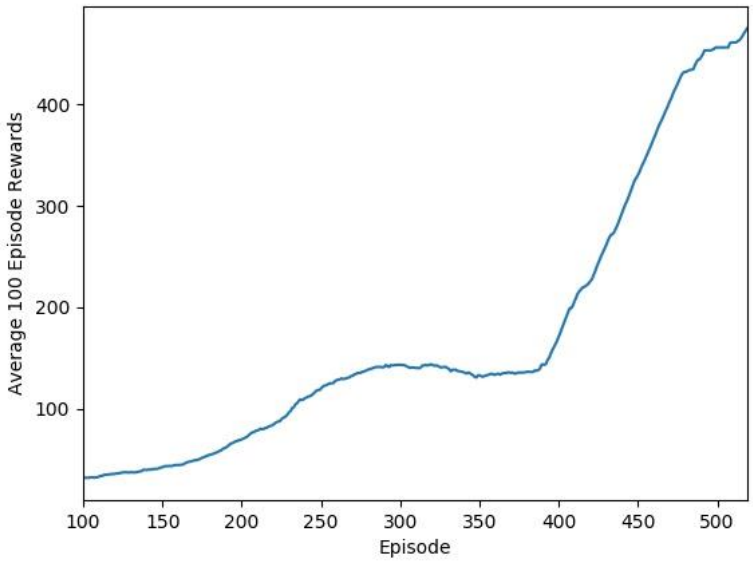
Hyperparameters:

- **Learning Rate - Actor**: 0.0007
- **Learning Rate - Critic**: 0.006
- **Discount Factor**: 0.99

Results:

* TensorBoard graphs are added in the output folder of each agent. Here I present manually designed plots since TensorBoard had problems I couldn't overcome.

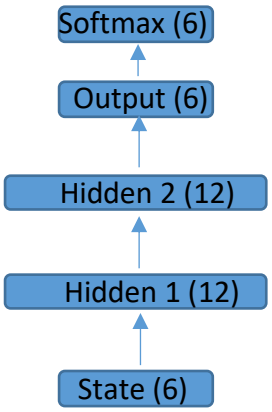




- Episodes until solved: **518** episodes
- Time until solved: **270.74** [sec]

2. **Acrobot Agent** – *acrobot_expert.py*

Actor's NN architecture:



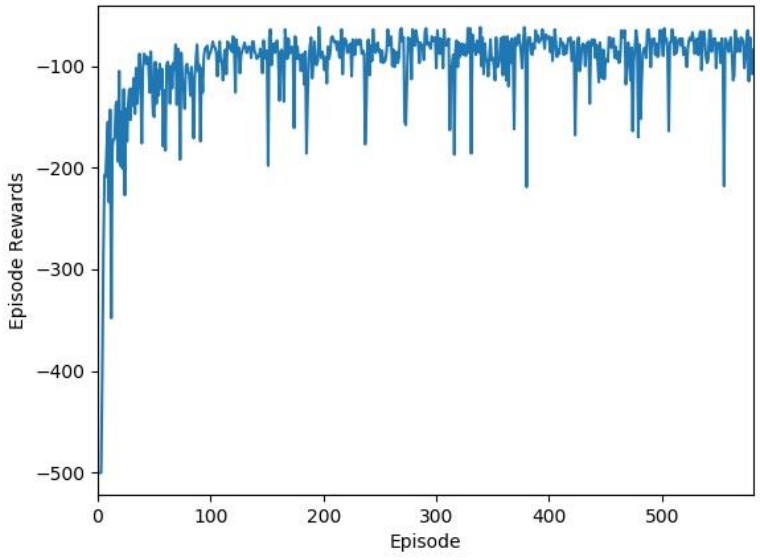
Settings:

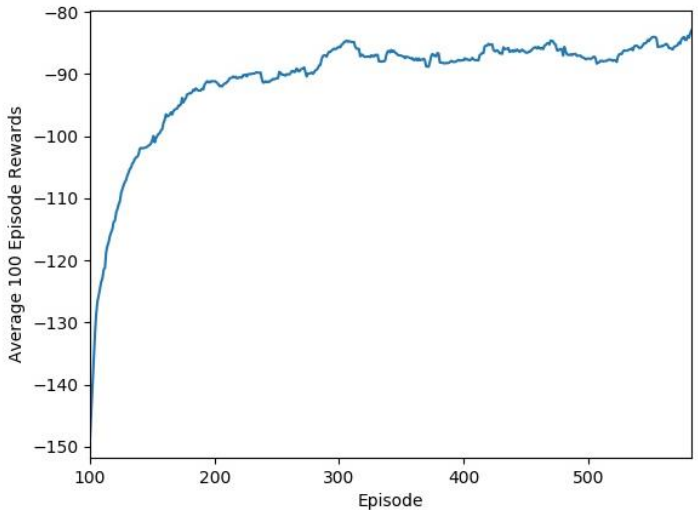
- **Actions probability distribution**: Softmax
- **Loss function**: Softmax Cross-entropy + **weight decay regularization**
- **Activation Function**: ReLU
- **Optimizer**: Adam
- **Number of original actions**: 3 (0, 1, 2)
- **Actions Dictionary**: {**0**: 0, **1**: 0, **2**: 1, **3**: 1, **4**: 2, **5**: 2}
- **Critic Network**: 2 hidden layers of size 20 and MSE loss
- **Solved definition**: Average reward of -83 for 100 consecutive episodes (defined by me…)

Hyperparameters:

- **Learning Rate - Actor**: 0.0008
- **Learning Rate - Critic**: 0.002
- **Discount Factor**: 0.97

Results:



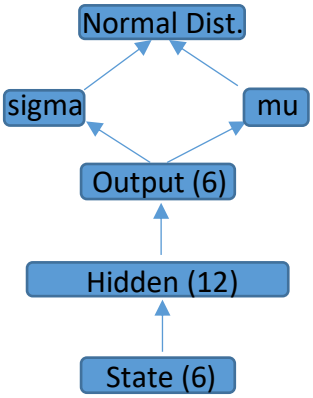


- Episodes until solved: **582** episodes
- Time until solved: **160.78** [sec]

3. **MountainCarContinuos Agent –** *mountain_expert.py*

Actor's NN architecture:

Since this environment has a continuous action space, I first tried to perform discretization of the space to 6 discrete actions (and more…) but it didn't work well. Instead, I took to neurons of the output layer of size 6 (the discrete constant actions size determined at the start of the assignment) and created a continuos normal distribution out of them:



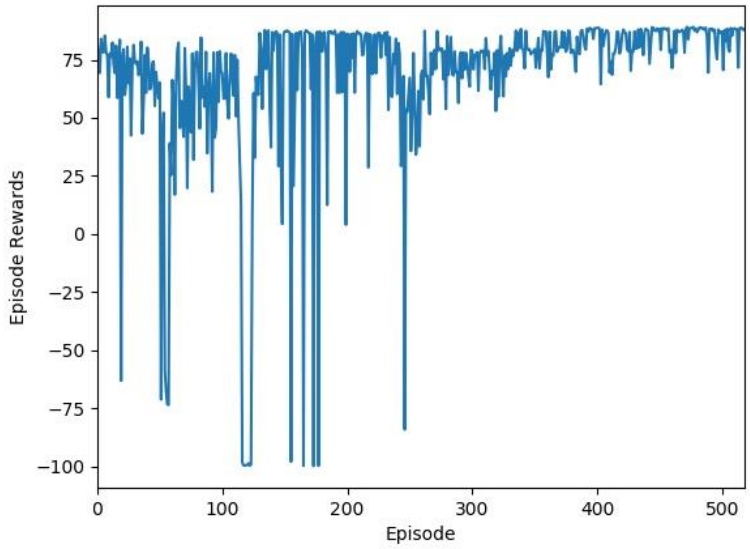
Settings:

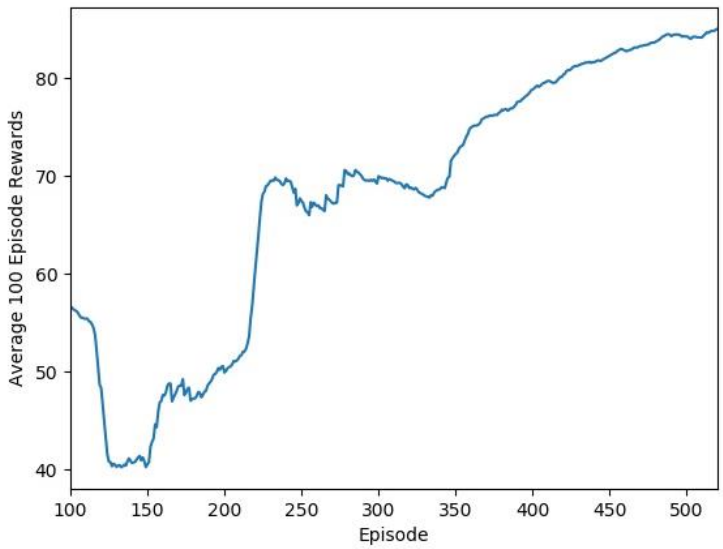
- **Actions probability distribution**: Normal distribution

- **Loss function**: Negative log-probability loss + **Entropy** (for exploration)
- **Activation Function**: eLU
- **Optimizer**: Adam
- **Number of original actions**: 1 (continuous)
- **Critic Network**: Same architecture but with output size of 1 and MSE loss
- **Solved definition**: Average reward of 90 for 100 consecutive episodes (didn't converge so stopped after average reward of 85)

Hyperparameters:
- **Learning Rate - Actor**: 0.0002
- **Learning Rate - Critic**: 0.001
- **Discount Factor**: 0.99

Results:





- Episodes until average reward 85: **520** episodes
- Time until solved: **280.56** [sec]

Section 2 – Fine-tune an existing model

In this section I restored an expert's NN that was trained on a specific task, reinitialized the weights of the output layer and retrained the network on a different task.

* **Because I do this assignment alone, you approved me to perform only one task in this section and one task in the following section.**

1. **Acrobot → CartPole** – *acrobot_cartpole.py*

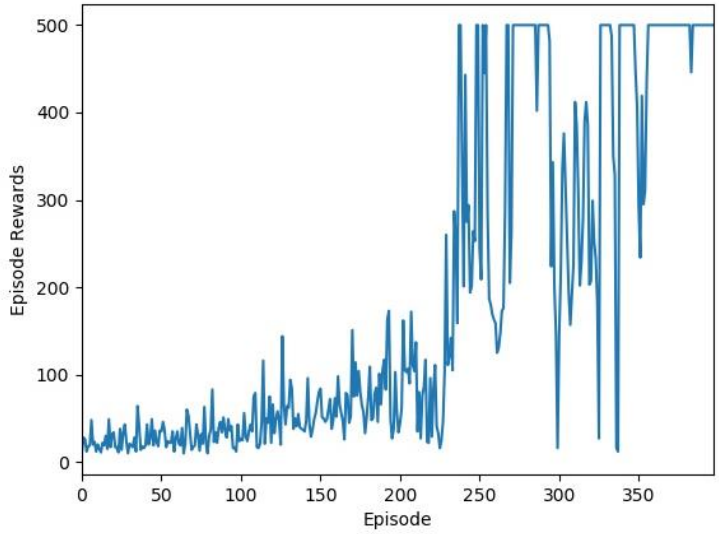
   NN Architecture: same as in Acrobot's expert agent.

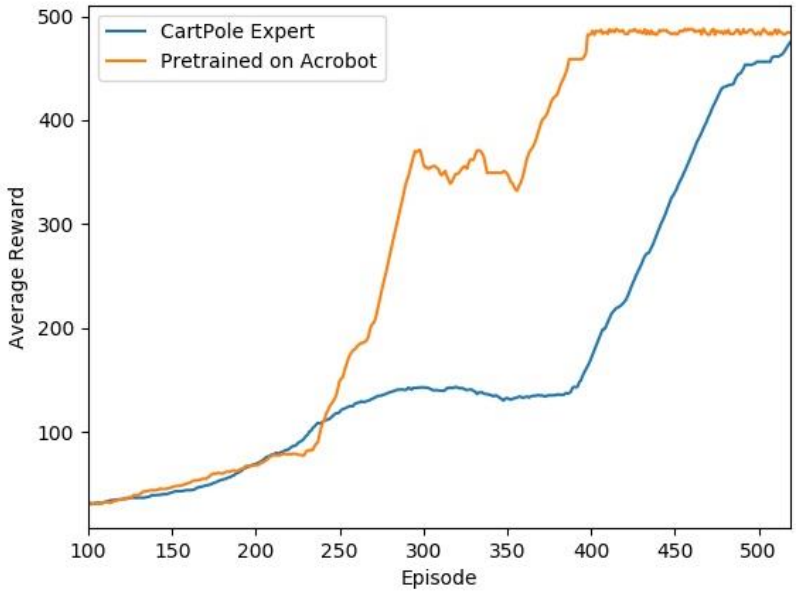
   Settings: same settings as in CartPole's expert agent

   Hyperparameters:

   - **Learning Rate - Actor**: 0.0005
   - **Learning Rate - Critic**: 0.006
   - **Discount Factor**: 0.99

   Results & Comparison





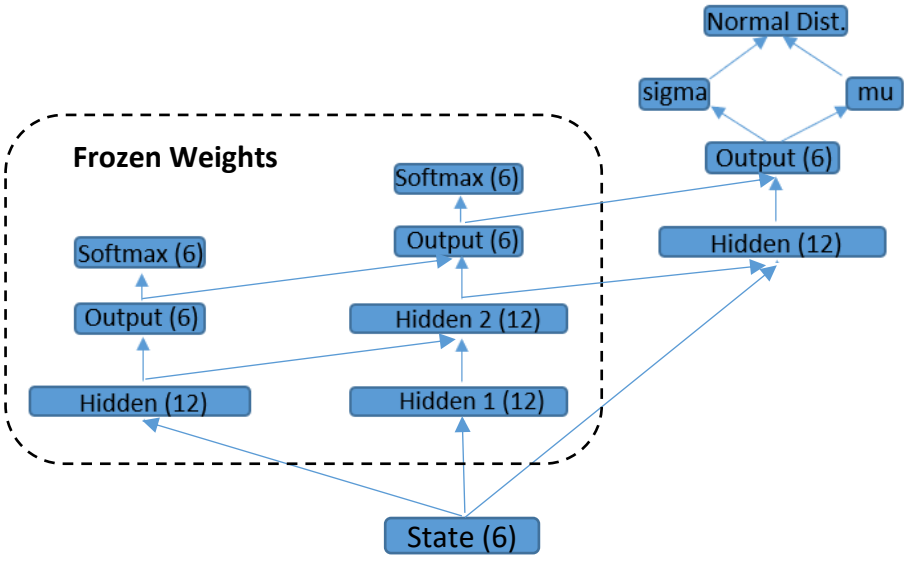
Agent	Solved at episode	Time to solved [sec]
CartPole Expert	518	270.74
CartPole with pre-trained Acrobot agent	**398**	**204.52**

Section 3 – Transfer Learning

In this section I restored two experts' NN that were trained on 2 different tasks, froze their layers and used them as inputs to a third NN that was trained on a third task.

1. **{Acrobot, CartPole} → MountainCar** – *mountaincar_progressive.py*

   Actor's NN Architecture:



Settings:

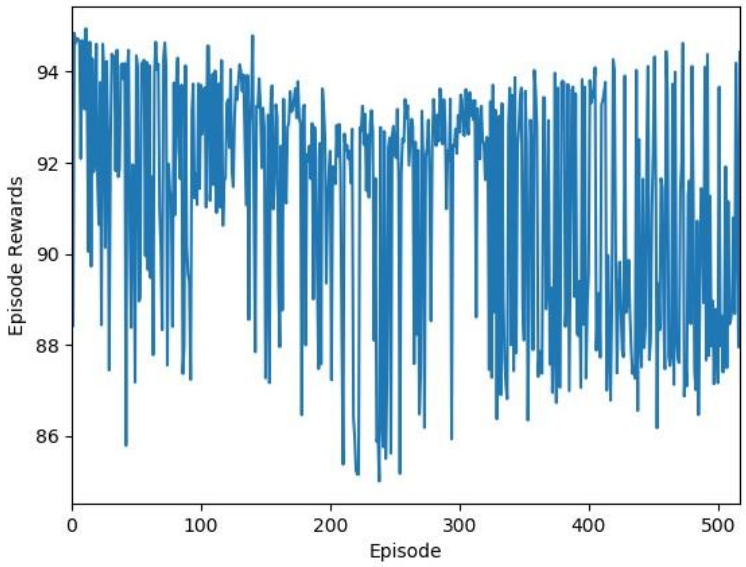
- **Number of original actions**: 3 (0, 1, 2)
- **Actions Dictionary**: {**0**: 0, **1**: 0, **2**: 1, **3**: 1, **4**: 2, **5**: 2}
- **Critic Network**: 2 hidden layers of size 12 and MSE loss
- **Solved definition**: Average reward of -83 for 100 consecutive episodes (defined by me…)

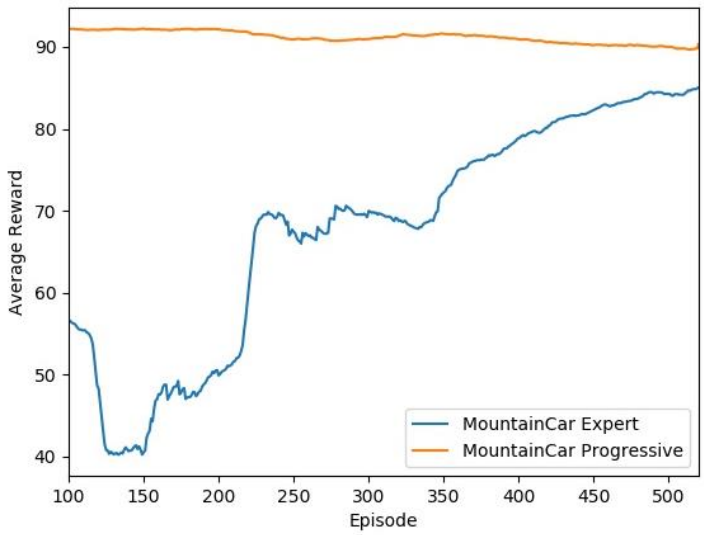
The rest of the settings are those of the two source expert agents and the target agent.

Hyperparameters:

- **Learning Rate - Actor**: 0.0002
- **Learning Rate - Critic**: 0.001
- **Discount Factor**: 0.99

Results & Comparison





Agent	Solved at episode	Time to solved [sec]
MountainCar Expert	520*	280.56*
MountainCar Progressive	**100**	**77.88**

* MountainCar Expert didn't reach average reward of 90 so defined solved at 85

Conclusions

1. Applying the Actor-Critic network of homework 2 on two new tasks was not straight-forward and required additional improvements. In the Acrobot task I added a regularization term to the loss function and in the MountainCar task I changed the probability distribution from Softmax to Normal distribution and added entropy to the loss to improve exploration.
2. Fine tuning of a pre-trained agent on a new task proved a difficult task but was successful eventually with faster convergence than just training an expert agent. Interestingly, the pre-trained agent started off the training in about the same place as the randomly initialized agent but had a sudden jump in performance early on in the fine-tuning process.
3. The progressive network achieved huge success, when visibly starting the training process with already very high performance. Even though the progressive network is much heavier, its convergence time was significantly shorter than that of the expert agent. Disclaimer: the performance of the progressive network was heavily influenced by the randomly initialized weights of the target network.