# Multi-Task Learning on classic control tasks with Deep Q Learning

Bachelor's Project Thesis

Thijs van der Laan, s3986721, j.t.van.der.laan@student.rug.nl,
Supervisor: dr. Matthia Sabatelli

**Abstract:** In Reinforcement Learning, an agent is often trained in only one environment. Consequently, it becomes fitted to this environment and is therefore ineffective in other environments. Human learning capabilities, on the contrary, show that learning multiple tasks at once is possible and can even be beneficial in terms of learning efficiency and time. Simultaneously learning multiple tasks is called Multi-Task Learning. This paper investigates whether a Deep Q-Learning agent using a multilayer perceptron as a function approximator could also benefit from Multi-Task Learning by simultaneously training it on the classic control problems Acrobot, Cartpole, and Mountaincar. Ultimately, we find that an agent can be trained to solve Acrobot and Cartpole comparably to a traditionally trained agent. We observe varying success between different hyperparameter configurations of epsilon-values, episodes between switching environments, or usage of regularizers. However, an agent trained in three environments shows less evidence of successful training in all environments.

## 1 Introduction

Traditionally in Machine Learning, a model is trained to succeed at one task. Whether this is image classification, object recognition or a reinforcement learning problem, the training of the model is limited to one task [Teh et al., 2017]. A major drawback of this approach is that the model becomes overspecialized to one specific problem and therefore will be unable to perform in another setting.

Humans, however, can learn multiple tasks simultaneously and use the learning process of one task to benefit learning another task. As illustrated by Zhang and Yang [2017], the skill of learning tennis can prove to be beneficial when also learning squash. When learned at the same time, humans will ultimately be able to play both tennis and squash. Additionally, learning these skills simultaneously could be beneficial for the learning speed.

Multi-Task Learning (MTL) [Caruana, 1997], is an approach in machine learning that aims to replicate this way of learning by training a model on several tasks at once. The goal is to enable one model to perform well on multiple tasks. By training on multiple tasks simultaneously, knowledge from one task can be used while learning other tasks potentially enhancing the speed of learning and the performance after training. Most importantly, the trained model is more versatile as it is able to perform in multiple tasks [Zhang and Yang, 2017]. MTL should not be confused with Transfer Learning (TL) [Pan and Yang, 2010, Zhuang et al., 2019], in which a model is pre-trained in a setting before being fine-tuned on the target task, effectively using learned information from the pre-training to improve learning on the target task.

MTL can be a workaround in situations of data sparsity or lack of computational resources [Sabatelli and Geurts, 2021]. When tasks have insufficient data to train independently, MTL enables a model to be trained on using multiple tasks, effectively combining the sparse data sets to guarantee good performance after training [Zhang and Yang, 2017, Mormont et al., 2021, Parisotto et al., 2016]. Furthermore, in fields such as Natural language Processing (NLP) where sufficient data is present, deep MTL models can achieve better performance than single-task models because they are less likely to overfit [Collobert and Weston, 2008, Zhang and

Yang, 2017]. Additionally, when a model has been trained on enough base tasks with use of MTL, it could become a promising pre-training candidate for TL [Parisotto et al., 2016, Sabatelli and Geurts, 2021].

Most success with MTL is achieved in combination with supervised learning [Liao and Carin, 2006, Gong et al., 2014], but also Reinforcement Learning (RL) can benefit from MTL [Wilson et al., 2007, Lazaric and Ghavamzadeh, 2010]. Recent work on MTL for object localization in combination with RL showed that MTL methods achieved higher average precision with fewer search steps in the state space [Wang et al., 2019]. In addition to this, Deshmukh et al. [2017] used MTL on a variation of the multi-armed bandit problem. The agent had to leverage similarities of different arms in order to better predict rewards. While their multi-task approach outperforms other methods, there remains room for improvement. Another study, [Sodhani et al., 2021], shows that RL agents, when given access to so called *metadata*, can achieve state-of-the-art results on challenging multi-task benchmarks.

While these results are promising, issues could arise when MTL is combined with RL. Instead of improving the training results, the multitask approach can affect the learning negatively on an individual task level, resulting in mediocre performance across all tasks, or specific tasks being dominant over others [Parisotto et al., 2016, Teh et al., 2017]. It is believed that this is a consequence of interference between the various policies, reward scales that differ, and complications in learning value functions in a stable way [Rusu et al., 2016]. Similar issues arise when RL is combined with Transfer Learning [Sabatelli and Geurts, 2021]. To obtain desirable results when two or more tasks are involved during training, additional techniques seem to be necessary.

Teh et al. [2017], for example, propose a new approach on shared network training called "Distral", which allows task-specific policies to contribute to a shared policy by using a variation of the distillation technique [Rusu et al., 2016]. In essence, knowledge from one task is 'distilled' into the shared policy and later to the other task-specific policies and vice versa.

While these more complex methods show promising performance, extra complexity is often Sundesirable and raises the question whether MTL can be successful in RL without additional techniques. Ideally, a straightforward and simple approach to MTL in combination with RL would be found, while sticking as close to the original RL-algorithms as possible.

In this study, we seek to find out whether there is a difference in task performance of a Deep Q-Learning-agent on *classic control problems* after training between a multi-task model and a single-task model. Both dual- and triple-task models are attempted solely with different hyperparameter configurations to keep the approach similar to the single-task 'single-task' models. Additionally, the different learning trajectories will also be analysed to determine whether MTL also proves to be beneficial for the speed of learning these tasks.

We hypothesise that there is a difference in performance after training between the multi-task model and the single-task model, but with additional differences between hyperparameters configurations as well. Issues such as the ones presented in Parisotto et al. [2016], Rusu et al. [2016], Teh et al. [2017] could arise.

# 2 Background

## 2.1 Preliminaries

In Reinforcement Learning (RL) artificial agents are tasked to learn an optimal behaviour in a specified setting by interacting with an environment and receiving rewards based on this interaction [Sutton and Barto, 2018].

An essential concept to model uncertain environments in which these agents take decisions are Markov Decision Processes (MDPs). A definition of a discrete MDP as in Sutton and Barto [2018] can be the following: An agent interacts with an environment during episodes with a specified length $T$, which is a number of discrete time steps. Therefore, at each time step $t$, a representation of the state of the environment, $s_t \in \mathcal{S}$ is given to the agent. It is then up to the agent to choose an action, $a_t \in \mathcal{A}$ to perform. As a consequence of this action, the agent receives a numerical reward, $r_{t+1}$, and subsequently has arrived in a new state, $s_{t+1}$. The trajectory repeats itself until the episode has finished or the goal has been reached. The Markov property, which states that the conditional distribution of state $s_{t+1}$ solely depends on $s_t$ is an essential assumption here.

Additionally, action selection of the agent is determined by a policy $\pi : \mathcal{S} \rightarrow \mathcal{A}$. $\pi$ is a map that contains which action to perform in each state.
To evaluate policies and the goodness of a state or a state-action pair, dynamic programming principles originating from Howard [1960] and Bellman [1966] are used. More specifically, the state value function $V(s_t)$ to assess the value of a state $s_t$ and state-action function $Q(s_t, a_t)$ to evaluate a state $s_t$ and its corresponding action $a_t$ [Sutton and Barto, 2018] are used.

When a policy $\pi$ is followed by the agent, the expected return when starting in a state $s$ can be computed with the *state-value function* $V^\pi$:

$$V^\pi(s) = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s, \pi\right]. \tag{2.1}$$

Here, $\gamma$ describes the so-called discount factor, which can be set between [0, 1]. When the discount factor is increased, the aim of the agents shifts from only considering rewards gained in the near future ($\gamma$ closer to 0) to maximizing the rewards obtained over a distant future ($\gamma$ closer to 1). To guarantee an effect of discounting, it is typically not set to 1.

In addition to the $V^\pi$, the expected return for taking an action $a$ in a state $s$ is given by the *state-action value function* $Q^\pi$, again assuming the agent is following policy $\pi$:

$$Q^\pi(s, a) = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s, a_t = a, \pi\right]. \tag{2.2}$$

In RL, training the agents consist of finding the optimal policy $\pi^*$, which maximizes the expected future reward from all possible states such that

$$\pi^*(s) = \underset{\pi}{\operatorname{argmax}}\, V^\pi(s). \tag{2.3}$$

A novel and central idea in RL is temporal-difference (TD) learning. Instead of updating the value function only after each episode, in TD this function is updated after each time step $t$. To do so, the reward of time step $t$ and an estimation of the value for state $s_{t+1}$ are used. With these values, the temporal difference target is computed [Sutton and Barto, 1990, Tesauro et al., 1995].

In classic RL, Tabular methods are often used. This works well for simple tasks, but many interesting tasks for RL come with an enormous state space. Saving, updating, and reading tables of these state spaces becomes a task that is computationally demanding and not feasible [Sutton and Barto, 2018]. A workaround for this computational problem is to use nonparametric function approximators [Busoniu et al., 2010], which nowadays often come in the form of Artificial Neural Networks (ANNs).

## 2.2 Deep Q Learning

A breakthrough in RL with help of TD Learning is the off-policy control algorithm Q-learning, which uses the state-action value $Q(s_t, a_t)$ Watkins and Dayan [1992]. It can be defined as:

$$\begin{aligned} Q(s_t, a_t) \leftarrow Q(a_t, a_t) + \alpha[ \\ r_{t+1} + \gamma \max_{a \in \mathcal{A}} Q(s_{t+1}, a) - Q(s_t, a_t)]. \end{aligned} \tag{2.4}$$

$r_{t+1} + \gamma \max_{a \in \mathcal{A}} Q(s_{t+1}, a)$ is the temporal difference target and $\alpha$ symbolizes the learning rate and defines how drastically the q-values should be updated.

During the training, actions are selected by either selecting the highest valued action from a Q-table in which q-values are stored or at random, often following an epsilon value or comparable exploration strategy. The obtained reward that follows from this action is then used to update the q-values in the table. This process repeats itself until a terminal state is reached.

Essentially, the learned action-value function Q approximates an optimal action-value function $Q^*$. This is done without considering the policy that the agent is following. When training is finished, the optimal policy can be found by always choosing the action that maximizes the Q-value in each state [Watkins, 1989, Watkins and Dayan, 1992, Sutton and Barto, 2018].

In this study, we use a Q-Learning variant with a function approximator. In this so-called Deep Q Learning, an ANN is trained as a function approximator for the state-action pair $Q(s, a)$. The function approximator replaces the Q-table. The parameterized neural network for the Q-value is defined as $\theta$ and it's corresponding loss function expressed in Mean Squared Error terms is defined as:

$$L_\theta = \mathbb{E}[(r_t + \gamma \max_{a \in \mathcal{A}} Q(s_{t+1}, a; \theta) - Q(s_t, a_t, ; \theta))^2]. \quad (2.5)$$

This loss function can then be minimized when training the Q function with the use of gradient descent.

The epsilon-greedy algorithm is used as exploration strategy (see 2.1 or 2.2). It consists of a linearly decreasing $\epsilon$-value that determines whether the agent should choose a random action (explore) or choose an action based on the q-value prediction of its function approximator (exploit).

In order to improve the stability of the training, the technique of "Experience Replay" [Lillicrap et al., 2015, Fedus et al., 2020] is added to the training routine. This method de-correlates the experiences used for training and enables the network to be trained on past episodes, which greatly benefits the stability of the training process of the ANN.

**The experience replay buffer** $D$, with size $N$, that is filled with experiences from the environment allows the agent to train on past episodes. These experiences are 5-tuples of the form $\langle s_t, a_t, s_{t+1}, r_t, done \rangle$, where *done* indicates whether or not an episode has finished after action $a_t$ was selected. When enough of these experiences have been saved in the buffer, a random sample of these can be taken to be used for training, allowing the model to benefit from past experiences multiple times. Moreover, the buffer is designed in a dequeue-like manner, which allows for a gradually increasing quality of the experiences. As training progresses, new experiences that led to higher rewards will enter the buffer, while old experiences with bad rewards from the exploration phase will be removed, allowing the quality of the samples to improve over time as well.

In this study, we will use replay buffers with sizes $N = 2000$ for *Acrobot* and *Cartpole* in all configurations, $N = 10.000$ for Mountaincar when trained on independently, and $N = 4000$ for *Mountaincar* when in the multi-task training cycle.

**The Target Network** is another concept that is used to increase stability during the training of Deep Reinforcement Learning agents. It involves a second ANN with the same architecture as the original model. The target network is used to estimate the targets that are necessary for computing the TD errors. Instead of updating the weights of this network each time a sample is taken from the Experience Replay Buffer, the Target Network acts as a stable network that is only updated periodically. Furthermore, instead of training the weights of this Target Network, the weights of the optimized network are simply copied. The target network is parameterized as $\theta^-$. The addition of this target network leads to a slight modification of the loss function from Equation 2.5:

$$L_\theta = \mathbb{E}_{\langle s_t, a_t, r_t, s_{t+1} \rangle \sim U(D)}[ (r_t + \gamma \max_{a \in \mathcal{A}} Q(s_{t+1}, a; \theta^-) - Q(s_t, a_t; \theta))^2] \quad (2.6)$$

The whole procedure of Deep Q learning is summarized in Algorithm 2.1

## 2.3 Multitask Deep Q Learning

In MTL, the RL agent is trained in multiple environments at once. To do so, it will cyclically alter these environments. The goal is to optimize the ANN used for function approximation so that it masters all tasks. We use a slightly modified version of the Deep Q-Learning algorithm. We define an RL agent that is trained on multiple environments $\mathcal{G}$. Because the environments have different action spaces $\mathcal{A}$, the ANN must have multiple output heads for all $g \in \mathcal{G}$. In essence, a model with corresponding output heads will be created for each environment, while keeping the hidden layers shared.

Additionally, the Experience Replay Buffer $D$ is extended to contain $N$ experiences for each environment. These experiences are environment-specific in terms of saving and sampling during the optimization of $\theta$. Moreover, multiple $\epsilon$-values are utilized. These can have different start values so that the balance between exploration and exploitation can be manipulated per unique environment. At last, a new hyperparameter has to be introduced that determines when training should continue to the next environment. We call this the *switch*-parameter. The environment alterations are done

**Algorithm 2.1** Deep Q Learning

```
Require: Experience Replay Dequeue $D$ with size $N$
Require: Q network with parameters $\theta$ and $\theta^-$
Require: total_e = 0
Require: $\mathcal{N} = 1000$
Require: $\epsilon_{min} = 0.01$
  for $e \in$ EPISODES do
    while $e\neg$ over do
      observe $s_t$
      select random action $a_t$ with probability $\epsilon$;
      Otherwise select $a_t = argmax_a Q_a(s_t, a, \theta)$;
      Execute $a_t$ and obtain $r_t$ and $s_{t+1}$;
      Store $e$ as $\langle s_t, a_t, s_{t+1}, r_t, done\rangle$ in $D$;
      if $\epsilon > \epsilon_{min}$ then
        $\epsilon * = 0.999$
      end if
      total_e += 1
      if total_e $\geq \mathcal{N}$ then
        sample minibatch of 64 $e$ from $D$
        Use gradient descent to minimize loss function Eq. 2.6 and update $\theta$
      end if
    end while
    Copy training parameters $\theta$ to target parameters $\theta^-$
  end for
```

in a cyclic way, where only the starting environment is determined at random. A summary of the Multitask Learning with Deep Q Learning can be found in Algorithm 2.2

**Algorithm 2.2** Multitask Learning with Deep Q Learning

```
Require: Experience Replay Dequeues $D_{g\in\mathcal{G}}$ with size $N$
Require: Q network with parameters $\theta_g$ and $\theta_g^-$, each with output heads $g \in \mathcal{G}$
Require: total_e = 0
Require: $\mathcal{N} = 1000$
Require: $\epsilon_{min} = 0.01$
Require: Random start environment $g_{current} \in \mathcal{G}$
  for $e \in$ EPISODES do
    if ($e$ mod $Switch$) == 0 then
      $g_{current} \Leftarrow g_{new}$
    end if
    while $e\neg$ over do
      observe $s_t$
      select random action $a_t$ with probability $\epsilon$;
      Otherwise select $a_t = argmax_a Q_a(s_t, a, \theta)$;
      Execute $a_t$ and obtain $r_t$ and $s_{t+1}$;
      Store $e$ as $\langle s_t, a_t, s_{t+1}, r_t, done\rangle$ in $D_g$;
      if $\epsilon_g > \epsilon_{min}$ then
        $\epsilon_g * = 0.999$
      end if
      total_e += 1
      if total_e $\geq \mathcal{N}$ then
        sample minibatch of 64 $e$ from $D_g$
        Use gradient descent to minimize loss function Eq. 2.6 and update $\theta_g$
      end if
    end while
    Copy training parameters $\theta_g$ to target parameters $\theta_g^-$
  end for
```

## 3 Methods

To test our hypothesis that there is a performance difference between multi-task models and their single-task models, we perform a series of experiments that shall be described below. These experiments will use several *classic control problems*, which are relatively computationally inexpensive and an important benchmark within the field of RL. These environments are provided by the Open-AI Gym package [Brockman et al., 2016]. Specifically, our RL agent will be trained on *Acrobot* [Sutton, 1996], *Cartpole* [Barto et al., 1983], and *Mountain-car* Moore [1990].

### 3.1 Environments

#### 3.1.1 Acrobot

The `Acrobot-v1` environment contains the *Acrobot* problem from Sutton [1996]. More specifically, it uses the implementation that is used by Geramifard et al. [2015]. The acrobot exists out of two joints and two links. The goal for the agent is to make the outer link swing above the horizontal line (see 3.1) in as little time as possible by controlling the torque on the joint that connects the links. This is done from a starting position where both links are facing downwards.

The observation space contains six numbers that

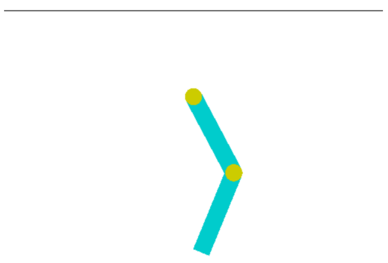


**Figure 3.1: Acrobot-v1**

provide information about the rotational joints and their angular velocities. Specifically: the angular velocity of the inner link $\theta_1$ (where 0 indicates it facing downwards), the relative angle of the outer link $\theta_2$ with respect to $\theta_1$ (where 0 indicates the link being in a straight line with the inner link), and the sine and cosine for both $\theta_1$ and $\theta_2$.

The action space consists of three possible actions. At each time step, the agent can choose to apply either positive or negative torque on the connection joint, or not apply torque at all. For each time step that the goal is not reached, a reward of -1 is obtained. When the goal state is reached, the reward for that time step is 0. The maximum duration of an episode is 500 time steps, making the minimum total reward -500. Acceptable rewards that display successful training are -150 and higher, which is higher than the single-task model.

### 3.1.2 Cartpole

The *Cartpole* problem contains a cart with an attached pole on top. The cart is controlled and can move in either the left or right direction. The attached pole is free to move in any direction, only remaining attached to the top of the cart. The goal is to keep the pole facing upwards by moving the cart.

The `CartPole-v0` environment follows the problem description from Barto et al. [1983]. An episode is over once the pole passes a 15-degree angle from the vertical starting position. The equations of motion of the cart-pole system are unknown. The Q-learning agent must move the cart in such a way that it avoids triggering this fail signal. The observation space is made of cart position, cart velocity, pole angle, and pole angular velocity. The action space consists of moving the cart left or right. Per timestep that the fail signal has not been given, the program receives a reward of +1. If the fail signal is given, the episode is over. The goal is to train the controller to achieve the highest possible reward of 200. In this version of the problem, cart friction is not taken into account. The single-task model achieves a perfect score of 200 in every episode, but stable rewards $> 175$ marks successful training.

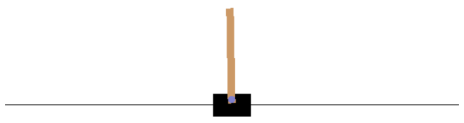


**Figure 3.2: CartPole-v0**

### 3.1.3 Mountaincar

The `MountainCar-v0` environment contains a car placed in the valley of a sinusoidal wave. An implementation following Moore [1990] will be used for our experiment. The goal is to make the car reach the red flag in as few time steps as possible.

The observation space consists of 2 floating-point numbers: the position of the car along the x-axis and the velocity of the car. At each time step, there are three possible actions that the agent can choose from. It can either accelerate the cart to the left, accelerate the cart to the right, or choose to not accelerate at all.

For every time step in which the flag is not reached, a reward of -1 is given. When the flag is reached, a reward of 0 is given and the episode is terminated. The maximum duration of an episode is 200 time steps, making the minimum reward -200. An acceptable reward after training is around -100, which is what our single-task model obtains.

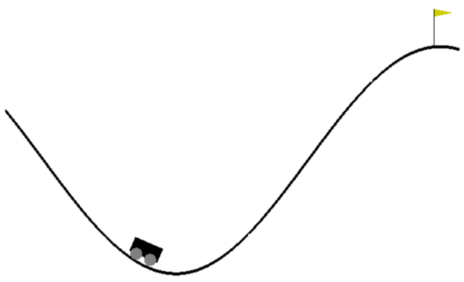


**Figure 3.3: MountainCar-v0**

## 3.2 Acrobot, Cartpole & Mountaincar Models

To verify the working of our multitask models. The performance of the trained models and average learning trajectory in each environment will be compared to that of a single-task model. To do so, one model is created for each environment that is solely trained on one of the three tasks.

All of these models are Multilayer Perceptrons (MLPs) with two hidden layers that use either ReLU non linearity ($f(x) = max(0, x)$) or linear ($f(x) = x$) as activation functions. The input and output layers are made to fit the state space and action space of each environment and the dense layers differ in size (see 3.4).

The Adam optimizer ([Kingma and Ba, 2014]) is used to optimize all MLPs. The epsilon-greedy algorithm is used as exploration strategy. The initial $\epsilon$ value is set to 0,5 for *Acrobot* and 1 for *Cartpole* and *Mountaincar*. This value is then linearly decreased to 0.01 (see 2.1 or 2.2). The discount factor is set to 0.99 and a learning rate of 0.001 is used.

## 3.3 Multitask Model

We examine two Multi-Task models. A Dual-Task model, and a Triple-Task model. Both follow the same architecture, only differing in the number of hidden layers and the number of output heads. For each environment in the training cycle, a shared hidden layer of 128 nodes is added to the model. Moreover, with each environment, a separate output head has to be added to the model to guarantee fitting action output. Essentially, this translates to 2 shared hidden layers followed by 2 output heads for the dual model and 3 of each for the triple model.

The multitask model, three tasks specifically, (see Figure 3.5) consists of three different models, that have shared input and hidden layers, but separate output heads. Because of differences in state space size, the states of *Cartpole* and *Mountaincar* are padded with 0's until equal in size to an *Acrobot* state. This way, all input can be fed to the model through one input layer.

To handle different action spaces, an output head is created for each environment. In addition to an environment-specific output layer, another unique dense layer is added to every head to allow more accurate action predictions. This architecture is loosely based on the model presented in Zhang and Yang [2017].

Again, the Adam optimizer [Kingma and Ba, 2014] is used to optimize the MLPs and the epsilon-greedy algorithm as an exploration strategy. However, the $\epsilon$ values are altered during the experiment and will be discussed more in-depth in Section 3.4. The discount factor and learning rate remain the same as for the single-task models.

## 3.4 Experiments

With these models, we perform a series of experiments, each with different hyper parameter configurations to see how performance is influenced. The following parameters are altered:

- Use of layer weight regularizers
- Amount of episodes before switching to another environment (see 2.2)
- The distribution of initial $\epsilon$ values

**Layer weight regularizers** are a tool in the Keras library [Chollet et al., 2015] that enables the user to apply penalties on layer activity during optimization. Essentially, the regularizer makes sure the individual weights of the MLP are not changed too drastically and therefore stabilizes the optimization process of the network. By allowing a more stable change of weights, regularizers could prove to be beneficial for an MTL model. Our model should be general enough to perform well in three tasks. Therefore, if the model weights are changed too drastically, a risk of 'forgetting' other tasks could emerge and performance would greatly fluctuate during training.

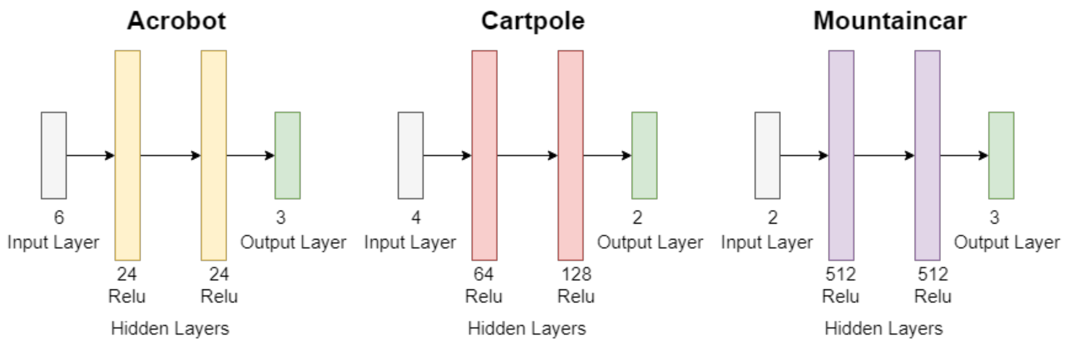


**Figure 3.4: The structure of the 3 MLPs used as control models. The activation-function of the hidden layers is Relu, while the output layers are linearly activated.**

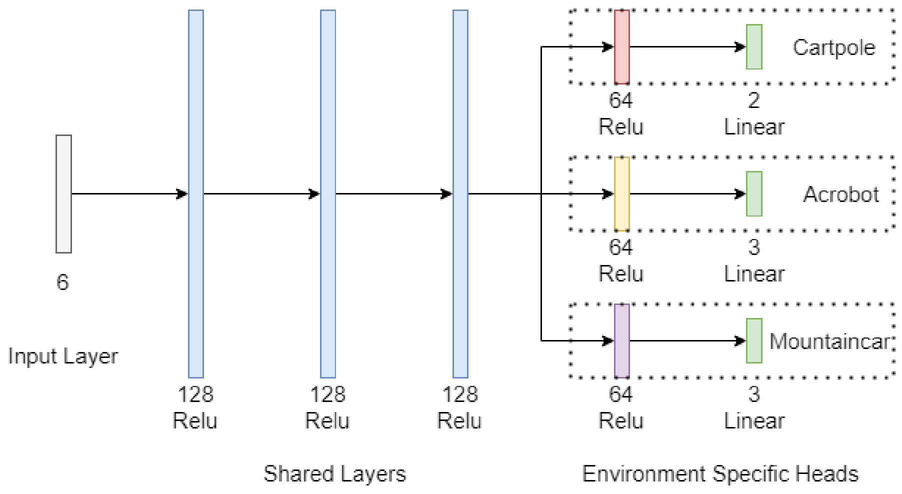


**Figure 3.5: The structure of the MLP used for the Multi-Task model. To suit the different action spaces, an output head for each environment has been created. The hidden layers are shared except for one environment-specific layer before the output.**

For our models, we use kernel regularization across the weights. L2 regularization [Farahmand et al., 2008], also known as 'Ridge Regularization', is applied instead of L1 regularization to avoid becoming too insensitive to some inputs. The model should be less sensitive to very subtle changes, but not completely ignore them either. For all configurations, an experiment with and without regularization is performed.

**Switch-episodes** are the number of episodes the MTL model will be trained on an environment before proceeding to the next environment. If this is done too fast, the model might not learn enough from these episodes to remember. If done too slow, on the other hand, the agent risks forgetting about what was learned earlier, which will result in a performance drop right after switching. To find an optimal length, three values will be examined. Environments are switched after 5, 10, or 20 episodes.

**$\epsilon$-values** are the essential part of the epsilon-greedy exploration strategy. Each environment in the loop will have its individual $\epsilon$, which will only be decreased while training on said environment. In theory, this leads to slower decreasing epsilons during training as these will only be decreased half or a third of the total training time. For the dual-task model, epsilon values $\langle 1; 0.5 \rangle$, $\langle 0.75; 0.75 \rangle$, $\langle 0.5; 0.5 \rangle$, $\langle 0.5; 1 \rangle$ will be considered.

For the triple-task model, these are $\langle 1; 0.5; 1\rangle$, $\langle 0.5; 1; 1\rangle$, $\langle 1; 1; 0,5\rangle$, $\langle 0.5; 0.5; 0.5\rangle$, $\langle 1; 1; 1\rangle$.

This leads to a series of experiments in which every configuration is trained for 1000 episodes for the dual-task model (500 episodes per environment) and 1200 episodes for the triple-task model (400 episodes per environment). Every training experiment is performed four times after which the average is taken. During training, the environment in which the model is trained changes in a cyclic manner starting with *Cartpole*, moving to *Acrobot* from here, and, in the case of the triple task model, at last, the *Mountaincar* environment. When all environments are seen, Cartpole is selected again. The starting environment is randomly initiated.

To verify proper training of the models, a 100-episode greedy run of every saved model will be performed. The averaged values of these will be tested for significant differences and compared to the single-task models.

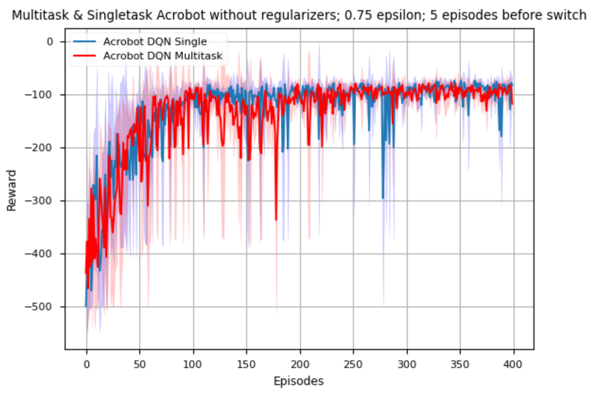
## 4 Results

Graphs of all results can be found in Appendix C and D; Subtle differences between different configurations can be observed, but, in general, the influence of hyperparameter follows the same trend. The exploitation run results are statistically compared to the exploitation data of the teacher models. Results of these tests can be found in Appendix A.

### 4.1 Dual-Task model

The results obtained with the Dual-Task model on *Acrobot* and *Cartpole* show considerable success in achieving desired performance levels on both tasks after training.

Figure 4.1 shows that the learning curve of the Dual-Task model on *Acrobot* is very comparable to that of the teacher model trained on *Acrobot*. *Cartpole*, however, falls short. When running the trained models from this Noreg-0,75-0,75-5eps configuration greedily for 100 episodes, we obtain an average reward of -90.1±17.2 for *Acrobot* and 176.1± for *Cartpole*. Both these are significantly different from the control averages of the teacher models (see Appendix A), with the *Acrobot*-score being higher and the *Cartpole*-score being lower. Nevertheless, both scores are close to those of the teacher model.



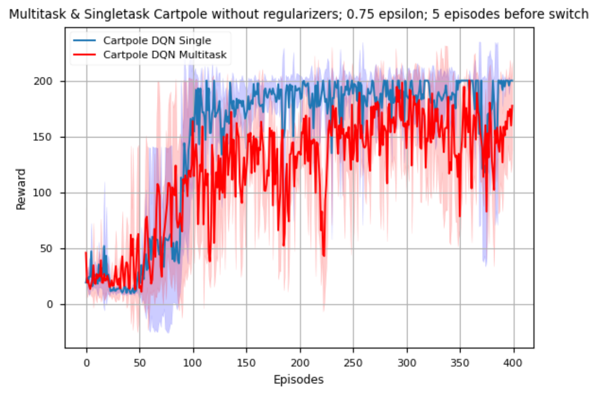


**Figure 4.1: Training with 5 episodes in between switching environments. Average learning trajectories of the Dual-Task model on *Acrobot* and *Cartpole* when training for 1000 epochs total. The average was taken over 4 rounds of training.**

Furthermore, when comparing Figure 4.2 with Figure 4.1, less successful training can be observed when the episodes between environments are increased to 20. The trajectory of *Acrobot* has more fluctuations with multiple downwards peaks. In Figure 4.1 only one serious peak can be observed. *Cartpole*'s trajectory also has more drops when compared, but recovers quickly back to desired performance. However, it never manages to remain on this level for more than around 15 episodes.

Similar trends, where training performance decreases as the episodes before switching increase, are observed across all configurations of the Dual-Task model (see Appendix C).

From Figure 4.3, an observation on the use of L2

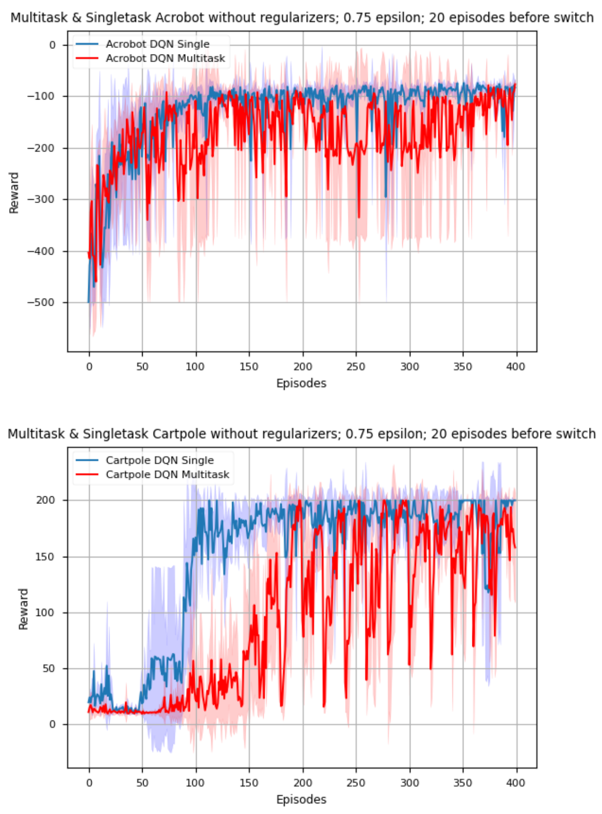


**Figure 4.2: Training with 20 episodes in between switching environments. Average learning trajectories of the Dual-Task model on *Acrobot* and *Cartpole* when training for 1000 epochs total. The average was taken over 4 rounds of training.**

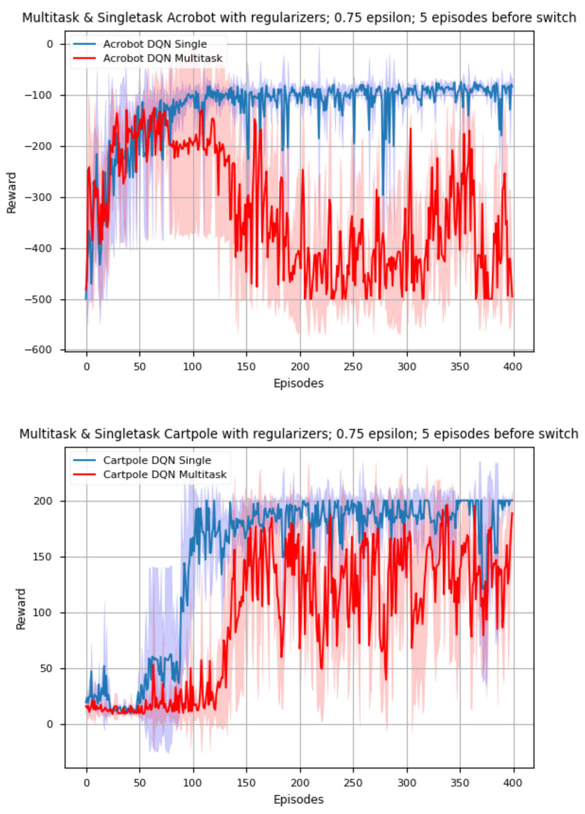


**Figure 4.3: Training with regularizers. Average learning trajectories of the Dual-Task model on *Acrobot* and *Cartpole* when training for 1000 epochs total. The average was taken over 4 rounds of training.**

kernel regularizers can be made when compared to Figure 4.1. We see an overall decrease in rewards for *Acrobot* at around 150 episodes. From this point on, the rewards obtained on *Cartpole* seem to increase. Moreover, when compared, the regularized model has more fluctuated rewards. This is especially accentuated in the graph for *Acrobot*. A decrease in performance when regularizers are used is visible for all configurations of the Dual-Task model (see Appendix C).

## 4.2 Triple-Task model

The results obtained with the Triple-Task model on *Acrobot*, *Cartpole* and *Mountaincar* do not display successful training on all three tasks. In Figure 4.4 it is possible to see that *Acrobot* reaches a performance level somewhat similar to the teacher model and follows its training pattern. However, from the exploitation data a mean of -366±19.5 is obtained, which is statistically different from the control mean of -145.8±51.6 and below acceptable performance. *Cartpole* is able to reach the maximum reward of 200, but fails to contain this learning and falls back to a reward below 100 throughout the training episodes. At last, *Mountaincar* shows small peaks of learning, after which periods of complete failure can be observed where the reward stays at -200. The amount of learning is the least for this task and the exploitation data confirms that no learning has occurred (see Appendix A).

Interestingly, however, training benefits do seem

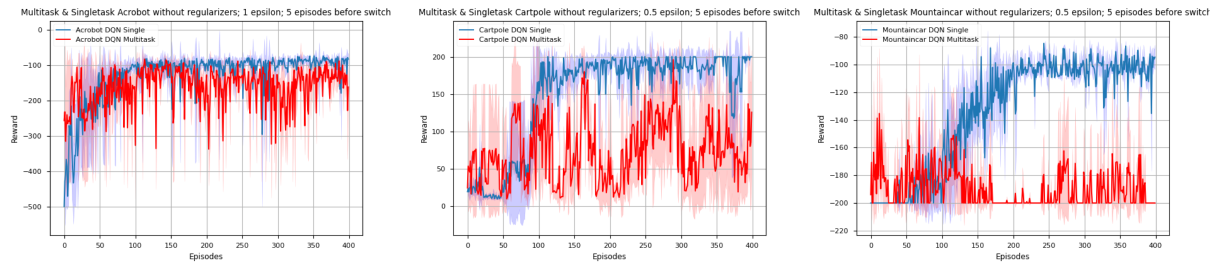


**Figure 4.4: Training with 5 episodes in between switching environments. Average learning trajectories of the Triple-Task model on *Acrobot*, *Cartpole* and *Mountaincar* when training for 1200 epochs total. The average was taken over 3 rounds of training.**

to occur in the first 50 episodes of the training. In Figure 4.4 it can be observed that for all 3 environments, the Triple-Task network obtains higher rewards faster than the teacher models in all environments. After these initial episodes, this phenomenon disappears and the obtained rewards decrease.

Even more so than with the Dual-Task models, we can observe an increase in fluctuated learning behavior when the amount of episodes in between switching environments is increased to 20. Specifically, in Figure 4.5 it can be observed that the obtained rewards in *Acrobot* keep reaching the same heights as in Figure 4.4 but on the other hand, also drops down to -400 at around 260 episodes. The results of *Cartpole* show similar fluctuations when compared to Figure 4.4, but the peaks and drops are steeper. From the *Mountaincar* graph, we observe less learning overall, with fewer peaks. Additionally, all evidence of an initial learning benefit has disappeared.

Similar to the Dual-Task model, configurations with regularizers show a decreased learning performance. The data in Figure 4.6 show an overall decreased training performance for *Acrobot* where the obtained results are lower and the desirable reward of $> -150$ is achieved less often than configuration without the use of regularizers. *Cartpole* and *Mountaincar* show no learning at all. Only small increases in rewards can be observed, but for both environments, not even half the maximum reward is ever reached.

Similar trends as a result of hyperparameter changes can be observed for all configurations of the Triple-Task model (see Appendix D).

## 5 Discussion

Successfully applying Multi-Task Learning in a two-task setting seems to be achievable without additional techniques when sufficient alteration between tasks is present during the training phase and the ANN can freely change its weights while optimizing. Dual-Task models are able to beat the teacher *Acrobot* model in some configurations (see Appendix A), but the perfect score that the single-task model achieves in *Cartpole* is not matched in any configuration. The Dual-Task model can only come close.

The Triple-Task model does show successful Multi-Task Learning on three tasks. While *Acrobot* and *Cartpole* do show an attempt at learning in some configurations, the added third task of *Mountaincar* is never mastered and performance does not even come close to that of the teacher model.

In both dual and triple task configurations, the use of regularizers seems to be a deciding factor for successful training. More specifically, configurations that do not use regularizers perform better than ones that do. In some cases such as Figure 4.6 task domination as described by Teh et al. [2017] could possibly be observed. *Cartpole* and *Mountaincar* show little learning, but the *Acrobot* model at least shows a careful increase in reward over time. It could be that the network becomes too overfitted to *Acrobot* and is not flexible enough to incorporate the other two tasks again. The poor results obtained when using regularization could either mean that the optimization of the weights of the model should not be held back to keep maximum flexibility, or that further tuning of the regularizers is required. Additionally, trying out a com-

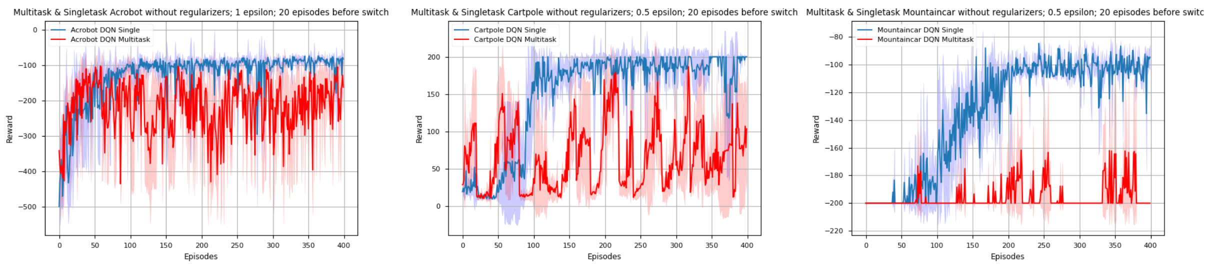


**Figure 4.5: Training with 20 episodes in between switching environments. Average learning trajectories of the Triple-Task model on *Acrobot*, *Cartpole* and *Mountaincar* when training for 1200 epochs total. The average was taken over 3 rounds of training.**

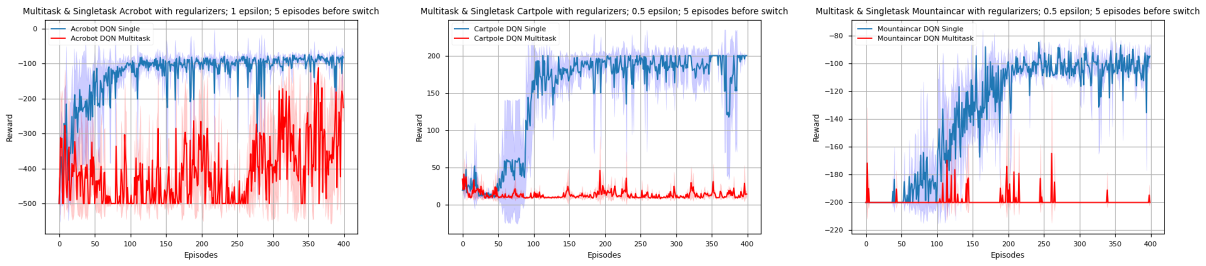


**Figure 4.6: Training with regularizers. Average learning trajectories of the Triple-Task model on *Acrobot*, *Cartpole* and *Mountaincar* when training for 1200 epochs total. The average was taken over 3 rounds of training.**

bination of both L1 and L2 regularization or using activation regularization instead of kernel regularization could provide us with different insights.

Another visible factor affecting the learning is the number of episodes before switching environments. More so for the Triple-Task models than for the Dual-Task models, a period of 20 episodes in between an environment switch seems to feed fluctuating rewards. The model likely spends too much time training other tasks sequentially, forgetting about the most previous task. Logically, this effect is graver for the Triple-Task model, where an episode switch of 20 translates to 40 episodes of training on other environments before restarting the environment cycle. For the Dual-Task model, this is only half. When the model eventually resumes training on this specific task again, the obtained rewards quickly return to previous levels, suggesting that only subtle changes to the weights are required to correct for this forgetting.

The initial epsilon values do not seem to affect the learning trajectories greatly. Other than subtle differences across configurations (See Appendix C and D), they do not seem to be the deciding factor for performance on all tasks.

These findings, especially the mediocre performance of the Triple-Task model, are in line with the possible drawbacks and issues for RL in combination with MTL as described by Rusu et al. [2016], Parisotto et al. [2016], Teh et al. [2017]. We observe either equal or worse performance of our multi-task models when compared to the teacher models as the results presented in Rusu et al. [2016]. Their results show an overall drop in learning across all 3 tasks, with *Acrobot* showing the most evidence of learning and *Mountaincar* often not at all.

However, in the trajectories of the Triple-Task model, there is some evidence of MTL being beneficial during training. As displayed in Figure 4.4, the multitask model outperforms the single-task model for all 3 tasks in the first 50 episodes. Likely, knowledge gained in one environment is used while training in another environment to accelerate learning speed and obtain higher rewards faster as described by Zhang and Yang [2017]. The question that remains is why the rewards drop after these initial episodes instead of continuing this trend.

Additional limitations to our methods could be that the Multi-Task models possibly require a longer training time before the network converges. Especially for the Triple-Task model, the number of training episodes per environment decreased to 400 as compared to the 500 for the teacher model. This might simply not be enough time for our model to find general patterns across all tasks and combine those in a useful way.

Alternatively, the ANN used for our Multi-Task model may not have been sufficiently large to generalize all 3 tasks. Considering that the teacher *Mountaincar* model has two dense layers with 512 nodes each, 3 layers with 128 nodes and one task-specific layer with 68 nodes might not be enough.

Future research could look further into this and vary in network architecture. Bigger, more complex networks, might result in better performance for all tasks. Several small tests with different architectures can be found in Appendix B.

Moreover, regularizers should be studied more in-depth in order to find out whether they could be a useful contribution to MTL. Although the results of this study do not point in that direction, this could be a consequence of our implementation rather than the properties of regularizers not being suitable for MTL.

At last, Deep Q-Learning in combination with Multi-Task Learning was explored in this thesis. Future research could combine MTL with other famous RL algorithms and explore if these are more suitable for training a multi-task network on Classic Control Tasks or other RL benchmarks.

## References

A.G. Barto, R.S. Sutton, and C.W. Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man and Cybernetics*, 13(5):834–846, 1983.

Richard Bellman. Dynamic programming. *Science*, 153(3731):34–37, 1966.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

Lucian Busoniu, Robert Babuska, Bart De Schutter, and Damien Ernst. *Reinforcement Learning and Dynamic Programming Using Function Approximators.* 01 2010. ISBN 1439821089. 10.1201/9781439821091.

Rich Caruana. Multitask learning. *Machine Learning*, 28:41–75, 1997.

Francois Chollet et al. Keras, 2015. URL `https://github.com/fchollet/keras`.

Ronan Collobert and Jason Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, page 160–167, New York, NY, USA, 2008. Association for Computing Machinery. ISBN 9781605582054. 10.1145/1390156.1390177. URL `https://doi.org/10.1145/1390156.1390177`.

Aniket Anand Deshmukh, Urun Dogan, and Clayton Scott. Multi-task learning for contextual bandits, 2017. URL `https://arxiv.org/abs/1705.08618`.

Amir Farahmand, Mohammad Ghavamzadeh, Shie Mannor, and Csaba Szepesvári. Regularized policy iteration. In D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, editors, *Advances in Neural Information Processing Systems*, volume 21. Curran Associates, Inc., 2008.

William Fedus, Prajit Ramachandran, Rishabh Agarwal, Yoshua Bengio, H. Larochelle, Mark Rowland, and Will Dabney. Revisiting fundamentals of experience replay. In *ICML*, 2020.

Alborz Geramifard, Christoph Dann, Robert H. Klein, William Dabney, and Jonathan P. How. Rlpy: A value-function-based reinforcement learning framework for education and research. *Journal of Machine Learning Research*, 16(46):1573–1578, 2015.

Pinghua Gong, Jiayu Zhou, and Jieping Ye. Efficient multi-task feature learning with calibration. *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 08 2014. 10.1145/2623330.2623641.

Ronald A Howard. Dynamic programming and markov processes. 1960.

Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 12 2014.

Alessandro Lazaric and Mohammad Ghavamzadeh. Bayesian multi-task reinforcement learning. In *ICML*, pages 599–606, 2010.

Xuejun Liao and Larry Carin. Radial basis function network for multi-task learning. *Advances in Neural Information Processing Systems*, 18:795, 01 2006.

Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning, 2015. URL `https://arxiv.org/abs/1509.02971`.

Andrew William Moore. Efficient memory-based learning for robot control. Technical report, University of Cambridge, 1990.

Romain Mormont, Pierre Geurts, and Raphael Maree. Multi-task pre-training of deep neural networks for digital pathology. *IEEE Journal of Biomedical and Health Informatics*, 25, feb 2021. 10.1109/jbhi.2020.2992878.

Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359, 2010. 10.1109/TKDE.2009.191.

Emilio Parisotto, Jimmy Lei Ba, and Ruslan Salakhutdinov. Actor-mimic: Deep multitask and transfer reinforcement learning, 2016.

Andrei A. Rusu, Sergio Gomez Colmenarejo, Caglar Gulcehre, Guillaume Desjardins, James Kirkpatrick, Razvan Pascanu, Volodymyr Mnih, Koray Kavukcuoglu, and Raia Hadsell. Policy distillation, 2016.

Matthia Sabatelli and Pierre Geurts. On the transferability of deep-q networks. *CoRR*, abs/2110.02639, 2021. URL `https://arxiv.org/abs/2110.02639`.

Shagun Sodhani, Amy Zhang, and Joelle Pineau. Multi-task reinforcement learning with context-based representations. *CoRR*, abs/2102.06177, 2021. URL `https://arxiv.org/abs/2102.06177`.

Richard Sutton. Generalization in reinforcement learning: Successful examples using sparse coarse coding. 08 1996.

Richard S Sutton and Andrew G Barto. Time-derivative models of pavlovian reinforcement. 1990.

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

Yee Whye Teh, Victor Bapst, Wojciech Marian Czarnecki, John Quan, James Kirkpatrick, Raia Hadsell, Nicolas Heess, and Razvan Pascanu. Distral: Robust multitask reinforcement learning. *CoRR*, abs/1707.04175, 2017. URL `http://arxiv.org/abs/1707.04175`.

Gerald Tesauro et al. Temporal difference learning and td-gammon. *Communications of the ACM*, 38(3):58–68, 1995.

Yan Wang, Lei Zhang, Lituan Wang, and Zizhou Wang. Multitask learning for object localization with deep reinforcement learning. *IEEE Transactions on Cognitive and Developmental Systems*, 11(4):573–580, 2019. 10.1109/TCDS.2018.2885813.

Christopher Watkins. Learning from delayed rewards. 1989.

Christopher J. C. H. Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8(3):279–292, May 1992. ISSN 1573-0565. 10.1007/BF00992698. URL `https://doi.org/10.1007/BF00992698`.

Aaron Wilson, Alan Fern, Soumya Ray, and Prasad Tadepalli. Multi-task reinforcement learning: a hierarchical bayesian approach. pages 1015–1022, 01 2007. 10.1145/1273496.1273624.

Yu Zhang and Qiang Yang. A survey on multi-task learning. *CoRR*, abs/1707.08114, 2017. URL `http://arxiv.org/abs/1707.08114`.

Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong, and Qing He. A comprehensive survey on transfer learning. *CoRR*, abs/1911.02685, 2019. URL `http://arxiv.org/abs/1911.02685`.

# A Appendix - Verification Run Results

| Environment | Switch | Epsilon (AB-CP) | Statistical Test Results |
|---|---|---|---|
| Acrobot | 5 | 0,5-1 | (*M*=-174.8, *SD*=62.9); t(99)=-3.5, p=0.0005 |
| Cartpole | | 0,5-1 | (*M*=139.4, *SD*=6.1); t(99)=-99.1, p=1.1e-170 |
| Acrobot | | 0,75-0,75 | (*M*=-90.1, *SD*=17.2); t(99)=10.2, p=7.0e-20 |
| Cartpole | | 0,75-0,75 | (*M*=176.1, *SD*=2.8); t(99)=-85.3, p=4.6e-158 |
| Acrobot | | 0,5-0,5 | (*M*=-225.5, *SD*=12.6); t(99)=-14.9, p=3.1e-34 |
| Cartpole | | 0,5-0,5 | (*M*=195.6, *SD*=15.9); t(99)=-2.7, p=0.007 |
| Acrobot | | 1-0,5 | (*M*=-177.0, *SD*=66.4); t(99)=-3.7, p= 0.0003 |
| Cartpole | | 1-0,5 | (*M*=166.7, *SD*=6.5); t(99)=-50.9, p=9.3e-116 |
| Acrobot | 10 | 0,5-1 | (*M*=-108.5, *SD*=42.5); t(99)=5.6, p=9.0e-8 |
| Cartpole | | 0,5-1 | (*M*=12-.2, *SD*=6.8); t(99)=-115.4, p=1.5e-183 |
| Acrobot | | 0,75-0,75 | (*M*=-99.4, *SD*=21.9); t(99)=8.2, p=2.3e-14 |
| Cartpole | | 0,75-0,75 | (*M*=192.6, *SD*=7.9); t(99)=-9.3, p=2.1e-17 |
| Acrobot | | 0,5-0,5 | (*M*=-140.7, *SD*=70.3); t(99)=0.6, p=0.56 |
| Cartpole | | 0,5-0,5 | (*M*=136.3, *SD*=6.9); t(99)=-91.3, p=9.7e-164 |
| Acrobot | | 1-0,5 | (*M*=-184.9, *SD*=90.3); t(99)=-3.7, p=0.0002 |
| Cartpole | | 1-0,5 | (*M*=197.9, *SD*=2.9); t(99)=-7.3, p=7.9e-12 |
| Acrobot | 20 | 0,5-1 | (*M*=-202.0, *SD*=58.1); t(99)=-7.2, p=1.3e-11 |
| Cartpole | | 0,5-1 | (*M*=186.5, *SD*=19.2); t(99)=-7.0, p=4.2e-11 |
| Acrobot | | 0,75-0,75 | (*M*=-232.2, *SD*=12.9); t(99)=-16.1, p=5.8e-38 |
| Cartpole | | 0,75-0,75 | (*M*=80.3, *SD*=5.6); t(99)=-211.4, p=3.6 |
| Acrobot | | 0,5-0,5 | (*M*=-93.1, *SD*=15.5); t(99)=9.7, p=1.5e-18 |
| Cartpole | | 0,5-0,5 | (*M*=106.7, *SD*=17.6); t(99)=-52.7, p=1.6e-118 |
| Acrobot | | 1-0,5 | (*M*=-112.0, *SD*=44.5); t(99)=4.9, p=1.7e-6 |
| Cartpole | | 1-0,5 | (*M*=72.9, *SD*=32.0); t(99)=-39.5, p=6.7e-96 |

**Table A.1: Dual Task model without regularizers. Statistical test consists of a unpaired two-tailed t-test with (*M*=-145.8, *SD*=51.6) for the Acrobot control model and (*M*=200, *SD*=0.0) for the Cartpole control model.**

| Environment | Switch | Epsilon (AB-CP) | Statistical Test Results |
|---|---|---|---|
| Acrobot | 5 | 0,5-1 | ($M$=-500, $SD$=0.0); t(99)=-68.3, p=1.5e-139 |
| Cartpole | | 0,5-1 | ($M$=145.8, $SD$=7.1); t(99)=-76.2, p=1.1e-148 |
| Acrobot | | 0,75-0,75 | ($M$=-500, $SD$=0.0); t(99)=-68.3, p=1.5e-139 |
| Cartpole | | 0,75-0,75 | ($M$=138.5, $SD$=19.8); t(99)=-30.9, p=1.3e-77 |
| Acrobot | | 0,5-0,5 | ($M$=-243.7, $SD$=59.5); t(99)=-12.4, p=2.1e-26 |
| Cartpole | | 0,5-0,5 | ($M$=148.5, $SD$=14.1); t(99)=-36.4, p=1.4 |
| Acrobot | | 1-0,5 | ($M$=-454.3, $SD$=42.6); t(99)=-45.9, p=1.9e-107 |
| Cartpole | | 1-0,5 | ($M$=117.9, $SD$=3.8); t(99)=-215.0, p=1.3e-236 |
| Acrobot | 10 | 0,5-1 | ($M$=-493.4, $SD$=22.7); t(99)=-61.3, p=10.0e-131 |
| Cartpole | | 0,5-1 | ($M$=65.7, $SD$=18.5); t(99)=-72.2, p=3.6 |
| Acrobot | | 0,75-0,75 | ($M$=-477.5, $SD$=46.9); t(99)=-47.3, p=7.4e-110 |
| Cartpole | | 0,75-0,75 | ($M$=152.1, $SD$=16.0); t(99)=-30.4, p=2.1e-76 |
| Acrobot | | 0,5-0,5 | ($M$=-500, $SD$=0); t(99)=-68.3, p=1.5e-139 |
| Cartpole | | 0,5-0,5 | ($M$=190.2, $SD$=5.0); t(99)=-19.6, p=3.0e-48 |
| Acrobot | | 1-0,5 | ($M$=-500, $SD$=0); t(99)=-68.3, p=1.5e-139 |
| Cartpole | | 1-0,5 | ($M$=172.6, $SD$=1.6); t(99)=-170.4, p=9.9e-217 |
| Acrobot | 20 | 0,5-1 | ($M$=-200.3, $SD$=75.8); t(99)=-5.9, p=1.4e-8 |
| Cartpole | | 0,5-1 | ($M$=78.5, $SD$=6.7); t(99)=-180.1, p=2.0e-221 |
| Acrobot | | 0,75-0,75 | ($M$=-427.0, $SD$=17.3); t(99)=-51.3, p=2.3e-116 |
| Cartpole | | 0,75-0,75 | ($M$=187.8, $SD$=9.2); t(99)=-12.1, p=9.7e-29 |
| Acrobot | | 0,5-0,5 | ($M$=-500, $SD$=0); t(99)=-68.3, p=1.5e-139 |
| Cartpole | | 0,5-0,5 | ($M$=198.0, $SD$=2.6); t(99)=-7.7, p=5.7e-13 |
| Acrobot | | 1-0,5 | ($M$=-251.7, $SD$=12.1); t(99)=-19.9, p=5.0e-49 |
| Cartpole | | 1-0,5 | ($M$=136.4, $SD$=0.2); t(99)=-2978.4, p=0.0 |

**Table A.2: Dual-Task model with regularizers. Statistical test consists of a unpaired two-tailed t-test with (*M*=-145.8, *SD*=51.6) for the Acrobot control model and (*M*=200, *SD*=0) for the Cartpole control model.**

| Environment | Switch | Epsilon (AB-CP-MC) | Statistical Test Results |
|---|---|---|---|
| Acrobot | 5 | 0,5-0,5-1 | ($M$=-382.3, $SD$=25.5); t(99)=-41.2, p=5.0e-99 |
| Cartpole | | 0,5-0,5-1 | ($M$=91.5, $SD$=6.3); t(99)=-170.8, p=6.5e-217 |
| Mountaincar | | 0,5-0,5-1 | ($M$=-200.0, $SD$=0); t(99)=-163.5, p=3.6e-213 |
| Acrobot | | 0,5-1-0,5 | ($M$=-240.1, $SD$=25.6); t(99)=-16.3, p=2.1e-38 |
| Cartpole | | 0,5-1-0,5 | ($M$=180.2, $SD$=4.7); t(99)=-42.1, p=9.5e-101 |
| Mountaincar | | 0,5-1-0,5 | ($M$=-195.1, $SD$=6.2); t(99)=-108.4, p=3.2e-178 |
| Acrobot | | 1-0,5-0,5 | ($M$=-366.8, $SD$=19.5); t(99)=-39.9, p=1.6e-96 |
| Cartpole | | 1-0,5-0,5 | ($M$=82.1, $SD$=5.9); t(99)=-200.4, p=1.4e-230 |
| Mountaincar | | 1-0,5-0,5 | ($M$=-200.0, $SD$=0); t(99)=-163.5, p=3.6e-213 |
| Acrobot | | 0,5-0,5-0,5 | ($M$=-240.1, $SD$=32.5); t(99)=-15.4, p=1.3e-35 |
| Cartpole | | 0,5-0,5-0,5 | ($M$=115.4, $SD$=5.6); t(99)=-151.0, p=2.2e-206 |
| Mountaincar | | 0,5-0,5-0,5 | ($M$=-200.0, $SD$=0); t(99)=-163.5, p=3.6e-213 |
| Acrobot | | 1-1-1 | ($M$=-326.7, $SD$=70.1); t(99)=-20.7, p=2.4e-51 |
| Cartpole | | 1-1-1 | ($M$=195.5, $SD$=11.5); t(99)=-3.9, p= 0.0001 |
| Mountaincar | | 1-1-1 | ($M$=-200.0, $SD$=0); t(99)=-163.5, p=3.6e-213 |
| Acrobot | 10 | 0,5-0,5-1 | ($M$=-242.7, $SD$=32.1); t(99)=-15.8, p=4.7e-37 |
| Cartpole | | 0,5-0,5-1 | ($M$=103.8, $SD$=29.7); t(99)=-32.3, p=9.2e-81 |
| Mountaincar | | 0,5-0,5-1 | ($M$=-199.6, $SD$=0.7); t(99)=-161.7, p=3.1e-212 |

| | | | |
|---|---|---|---|
| Acrobot | | 0,5-1-0,5 | (*M*=-367.2, *SD*=15.8); t(99)=-40.8, p=2.5e-98 |
| Cartpole | | 0,5-1-0,5 | (*M*=45.2, *SD*=17.5); t(99)=-88.1, p=8.3e-161 |
| Mountaincar | | 0,5-1-0,5 | (*M*=-200.0, *SD*=0); t(99)=-163.5, p=3.6e-213 |
| Acrobot | | 1-0,5-0,5 | (*M*=-376.4, *SD*=13.5); t(99)=-43.0, p=51.6 |
| Cartpole | | 1-0,5-0,5 | (*M*=118.1, *SD*=11.4); t(99)=-71.7, p=1.4e-143 |
| Mountaincar | | 1-0,5-0,5 | (*M*=-200.0, *SD*=0); t(99)=-163.5, p=3.6e-213 |
| Acrobot | | 0,5-0,5-0,5 | (*M*=-500.0, *SD*=0.0); t(99)=-68.3, p=1.5e-139 |
| Cartpole | | 0,5-0,5-0,5 | (*M*=34.7, *SD*=5.5); t(99)=-301.5, p=1.3e-265 |
| Mountaincar | | 0,5-0,5-0,5 | (*M*=-200.0, *SD*=0); t(99)=-163.5, p=3.6e-213 |
| Acrobot | | 1-1-1 | (*M*=-443.1, *SD*=56.5); t(99)=-38.7, p=3.4e-94 |
| Cartpole | | 1-1-1 | (*M*=102.7, *SD*=18.7); t(99)=-51.7, p=5.9e-117 |
| Mountaincar | | 1-1-1 | (*M*=-200.0, *SD*=0); t(99)=-163.5, p=3.6e-213 |
| Acrobot | 20 | 0,5-0,5-1 | (*M*=-316.3, *SD*=57.4); t(99)=-22.0, p=5.6e-55 |
| Cartpole | | 0,5-0,5-1 | (*M*=69.5, *SD*=7.1); t(99)=-183.4, p=5.4e-223 |
| Mountaincar | | 0,5-0,5-1 | (*M*=-200.0, *SD*=0); t(99)=-163.5, p=3.6e-213 |
| Acrobot | | 0,5-1-0,5 | (*M*=-348.1, *SD*=43.6); t(99)=-29.8, p=4.4e-7 |
| Cartpole | | 0,5-1-0,5 | (*M*=18.0, *SD*=3.7); t(99)=-495.0, p=3.7e-308 |
| Mountaincar | | 0,5-1-0,5 | (*M*=-200.0, *SD*=0); t(99)=-163.5, p=3.6e-213 |
| Acrobot | | 1-0,5-0,5 | (*M*=-185.0, *SD*=50.0); t(99)=-5.4, p=1.6e-7 |
| Cartpole | | 1-0,5-0,5 | (*M*=31.9, *SD*=3.7); t(99)=-450.0, p=5.7e-300 |
| Mountaincar | | 1-0,5-0,5 | (*M*=-200.0, *SD*=0); t(99)=-163.5, p=3.6e-213 |
| Acrobot | | 0,5-0,5-0,5 | (*M*=-500.0, *SD*=0.0); t(99)=,-68.3 p=1.5e-139 |
| Cartpole | | 0,5-0,5-0,5 | (*M*=16.7, *SD*=1.6); t(99)=-1140.6, p=0.0 |
| Mountaincar | | 0,5-0,5-0,5 | (*M*=-200.0, *SD*=0); t(99)=-163.5, p=3.6e-213 |
| Acrobot | | 1-1-1 | (*M*=-500.0, *SD*=0.0); t(99)=-68.3, p=1.5e-139 |
| Cartpole | | 1-1-1 | (*M*=157.3, *SD*=7.7); t(99)=-55.0, p=6.9e-122 |
| Mountaincar | | 1-1-1 | (*M*=-200.0, *SD*=0.0); t(99)=-163.5, p=3.6e-213 |

**Table A.3: Triple-Task model without regularizers. Statistical test consists of a unpaired two-tailed t-test with (*M*=-145.8, *SD*=51.6) for the Acrobot control model, (*M*=200, *SD*=0) for the Cartpole control model and (*M*=-100.8, *SD*=6.0) for the Mountaincar control model.**

| Environment | Switch | Epsilon (AB-CP-MC) | Statistical Test Results |
|---|---|---|---|
| Acrobot | 5 | 0,5-0,5-1 | (*M*=0379.9, *SD*=22.8); t(99)=-41.3, p=3.5e-99 |
| Cartpole | | 0,5-0,5-1 | (*M*=23.0, *SD*=5.8); t(99)=-306.0, p=7.4e-267 |
| Mountaincar | | 0,5-0,5-1 | (*M*=-200.0, *SD*=0.0); t(99)=-163.5, p=3.6e-213 |
| Acrobot | | 0,5-1-0,5 | (*M*=-470.4, *SD*=51.6); t(99)=-44.2, p=1.5e-104 |
| Cartpole | | 0,5-1-0,5 | (*M*=20.6, *SD*=4.8); t(99)=-368.0, p=1.0e-282 |
| Mountaincar | | 0,5-1-0,5 | (*M*=-200.0, *SD*=0.0); t(99)=-163.5, p=3.6e-213 |
| Acrobot | | 1-0,5-0,5 | (*M*=-500.0, *SD*=0.0); t(99)=-68.3, p=1.5e-139 |
| Cartpole | | 1-0,5-0,5 | (*M*=9.4, *SD*=0.5); t(99)=-4143.8, p=0.0 |
| Mountaincar | | 1-0,5-0,5 | (*M*=-200.0, *SD*=0.0); t(99)=-163.5, p=3.6e-213 |
| Acrobot | | 0,5-0,5-0,5 | (*M*=-424.1, *SD*=55.5); t(99)=-36.5, p=6.6e-90 |
| Cartpole | | 0,5-0,5-0,5 | (*M*=15.6, *SD*=2.6); t(99)=-700.3, p=0.0 |
| Mountaincar | | 0,5-0,5-0,5 | (*M*=-200.0, *SD*=0.0); t(99)=-163.5, p=3.6e-213 |
| Acrobot | | 1-1-1 | (*M*=-500.0, *SD*=0.0); t(99)=-68.3, p=1.5e-139 |
| Cartpole | | 1-1-1 | (*M*=22.2, *SD*=5.0); t(99)=-351.3, p=1.0e-278 |
| Mountaincar | | 1-1-1 | (*M*=-200.0, *SD*=0.0); t(99)=-163.5, p=3.6e-213 |
| Acrobot | 10 | 0,5-0,5-1 | (*M*=-306.3, *SD*=38.7); t(99)=-24.8, p=1.6e-62 |

| | | | |
|---|---|---|---|
| Cartpole | | 0,5-0,5-1 | (*M*=9.3, *SD*=0.4); t(99)=-4449.5, p=0.0 |
| Mountaincar | | 0,5-0,5-1 | (*M*=-200.0, *SD*=0.0); t(99)=-163.5, p=3.6e-213 |
| Acrobot | | 0,5-1-0,5 | (*M*=-374.9, *SD*=8.6); t(99)=-43.5, p=2.3e-103 |
| Cartpole | | 0,5-1-0,5 | (*M*=12.4, *SD*=1.8); t(99)=-1028.2, p=0.0 |
| Mountaincar | | 0,5-1-0,5 | (*M*=-200.0, *SD*=0.0); t(99)=-163.5, p=3.6e-213 |
| Acrobot | | 1-0,5-0,5 | (*M*=-497.9, *SD*=)6.4; t(99)=-67.3, p=2.0e-138 |
| Cartpole | | 1-0,5-0,5 | (*M*=12.3, *SD*=1.1); t(99)=-1571.1, p=0.0 |
| Mountaincar | | 1-0,5-0,5 | (*M*=-200.0, *SD*=0.0); t(99)=-163.5, p=3.6e-213 |
| Acrobot | | 0,5-0,5-0,5 | (*M*=-500.0, *SD*=0.0); t(99)=-68.3, p=1.5e-139 |
| Cartpole | | 0,5-0,5-0,5 | (*M*=24.2, *SD*=3.6); t(99)=-488.9, p=4.2e-307 |
| Mountaincar | | 0,5-0,5-0,5 | (*M*=-200.0, *SD*=0.0); t(99)=-163.5, p=3.6e-213 |
| Acrobot | | 1-1-1 | (*M*=-500.0, *SD*=0.0); t(99)=-68.3, p=1.5e-139 |
| Cartpole | | 1-1-1 | (*M*=42.0, *SD*=11.2); t(99)=-140.2, p=4.2e-200 |
| Mountaincar | | 1-1-1 | (*M*=-200.0, *SD*=0.0); t(99)=-163.5, p=3.6e-213 |
| Acrobot | 20 | 0,5-0,5-1 | (*M*=-159.0, *SD*=63.2); t(99)=-1.6, p=0.11 |
| Cartpole | | 0,5-0,5-1 | (*M*=9.3, *SD*=0.4); t(99)=-4417.2, p=0.0 |
| Mountaincar | | 0,5-0,5-1 | (*M*=-200.0, *SD*=0.0); t(99)=-163.5, p=3.6e-213 |
| Acrobot | | 0,5-1-0,5 | (*M*=-326.5, *SD*=64.3); t(99)=-21.8, p=1.6e-54 |
| Cartpole | | 0,5-1-0,5 | (*M*=13.6, *SD*=1.4); t(99)=-1283.1, p=0.0 |
| Mountaincar | | 0,5-1-0,5 | (*M*=-200.0, *SD*=0.0); t(99)=-163.5, p=3.6e-213 |
| Acrobot | | 1-0,5-0,5 | (*M*=-271.9, *SD*=26.3); t(99)=-21.6, p=4.3e-54 |
| Cartpole | | 1-0,5-0,5 | (*M*=9.4, *SD*=0.5); t(99)=-4141.5, p=0.0 |
| Mountaincar | | 1-0,5-0,5 | (*M*=-200.0, *SD*=0.0); t(99)=-163.5, p=3.6e-213 |
| Acrobot | | 0,5-0,5-0,5 | (*M*=-404.3, *SD*=20.2); t(99)=-46.4, p=2.7e-108 |
| Cartpole | | 0,5-0,5-0,5 | (*M*=28.9, *SD*=7.6); t(99)=-222.9, p=1.1e-239 |
| Mountaincar | | 0,5-0,5-0,5 | (*M*=-200.0, *SD*=0.0); t(99)=-163.5, p=3.6e-213 |
| Acrobot | | 1-1-1 | (*M*=-500.0, *SD*=0.0); t(99)=-68.3, p=1.5e-139 |
| Cartpole | | 1-1-1 | (*M*=16.4, *SD*=5.9); t(99)=-310.2, p=4.8e-268 |
| Mountaincar | | 1-1-1 | (*M*=-200.0, *SD*=0.0); t(99)=-163.5, p=3.6e-213 |

**Table A.4: Triple-Task model with regularizers. Statistical test consists of a unpaired two-tailed t-test with (*M*=-145.8, *SD*=51.6) for the Acrobot control model, (*M*=200, *SD*=0) for the Cartpole control model and (*M*=-100.8, *SD*=6.0) for the Mountaincar control model.**

## B Appendix - Various Alternative Model Architectures

As discussed before, a possible way to achieve better results for the Triple-Task model could be to modify the network structure. Considering the relatively big amount of nodes used in the layers of the Mountaincar single-task network compared to those of Acrobot and Cartpole and the poor results of our current Multi-Task model on all 3 tasks, we believe that an architecture with more nodes could be beneficial for the performance on the tasks.

Two attempts to Multi-Task Learning with slightly modified methods can be found below. These are by no means proper research, but the presented figures could aid in deciding on which modifications could be most useful in future research.

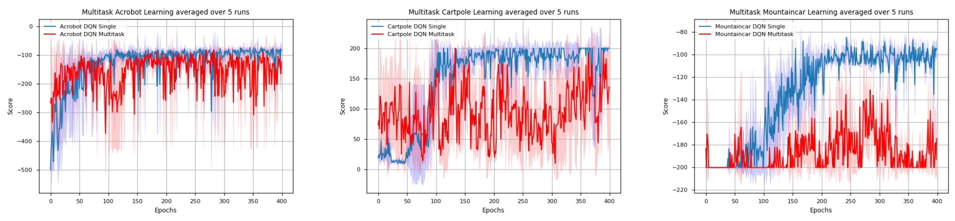


**Figure B.1: Model architecture where the task specific layer of Mountaincar is increases to fit 128 nodes in stead of 64. Without regularizers; 5 episode switch; epsilon values: AB: 0.5, CP: 0,5, MC: 1. Learning over 400 episodes averaged over 3 runs.**

Firstly, an increased number of nodes only in the task specific layer of Mountaincar, does seem to increase the rewards obtained in this environment as seen in Figure B.1. Here the dense layer with 64 nodes was doubled to 128. Further directions could incorporate an even higher number of nodes or even experiment with an extra task specific layer only for Mountaincar.

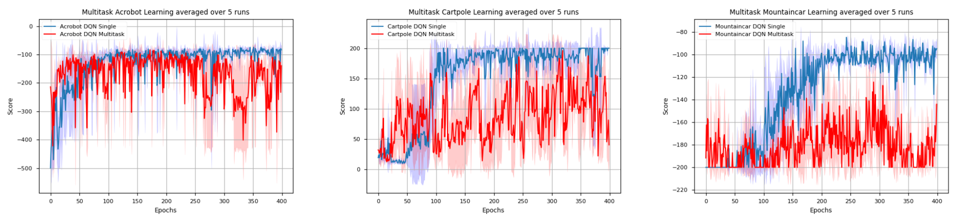


**Figure B.2: Model architecture where the shared hidden layers are increases to 256 nodes and the task specific layer for all tasks is increased to 128. Without regularizers; 5 episode switch; epsilon values: AB: 0.5, CP: 0,5, MC: 1. Learning over 400 episodes averaged over 3 runs.**

At last, when all nodes except for input and output layers are doubled, we also see an increase in reward in Mountaincar environment. Even though this is by no means a proper learning trajectory, these results do push us in the direction of attempting the same experiments, but simply with bigger, more complex networks.

## C Appendix - Dual-Task Model Results

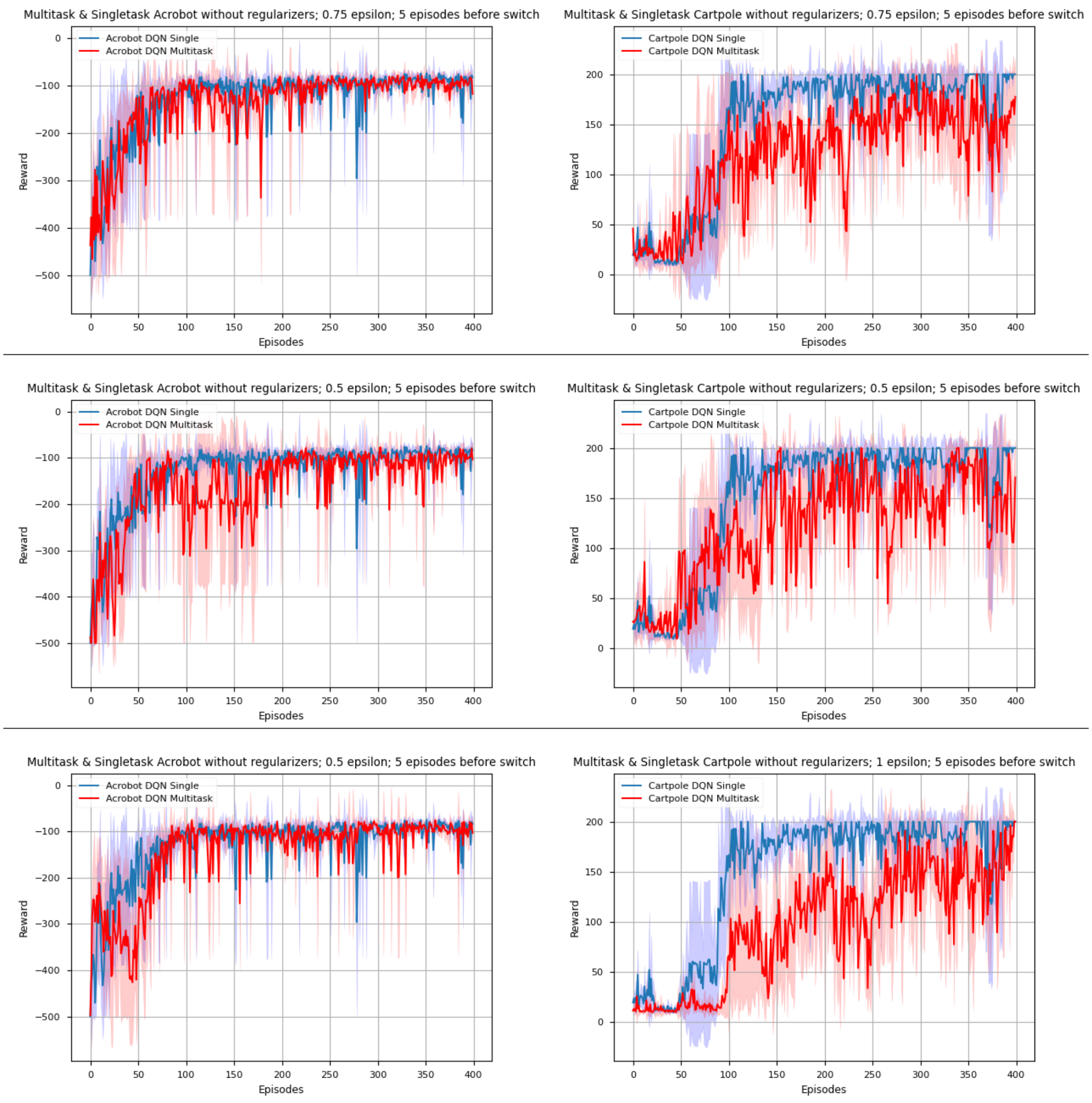


**Figure C.1: Dual-Task results without regularizers and 5 episodes in between switching episodes.**

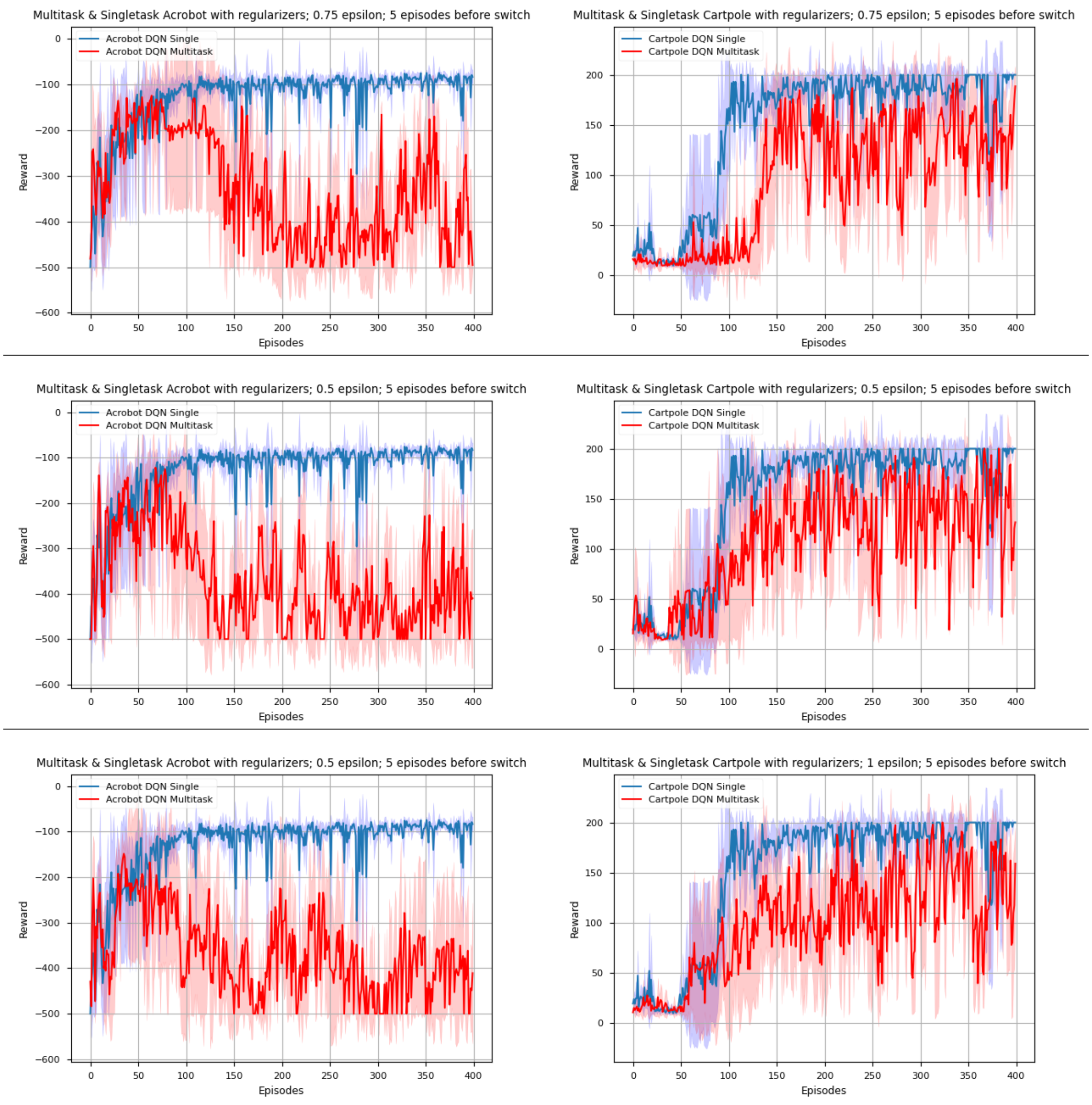


**Figure C.2: Dual-Task results with regularizers and 5 episodes in between switching episodes.**

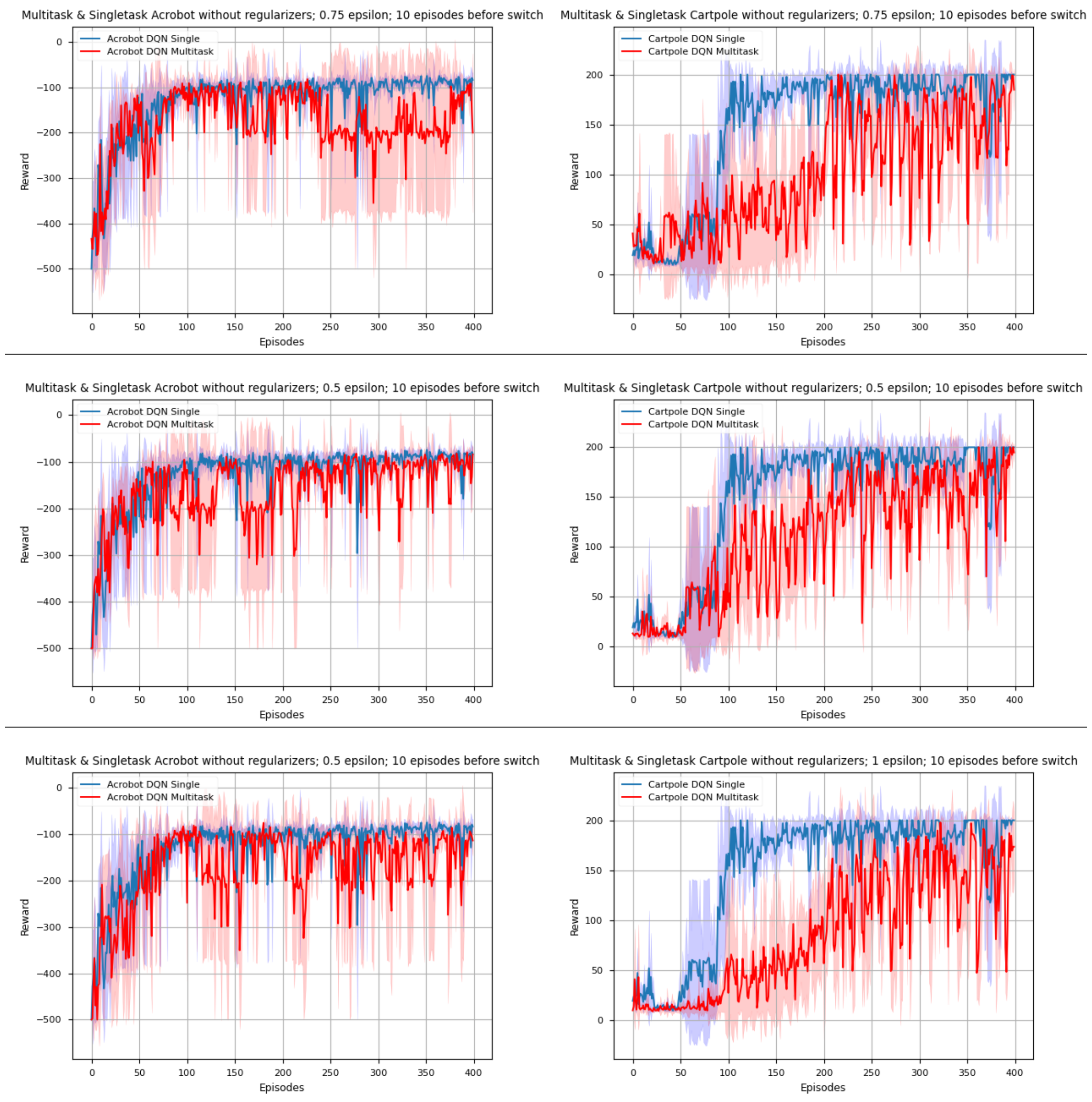


**Figure C.3: Dual-Task results without regularizers and 10 episodes in between switching episodes.**

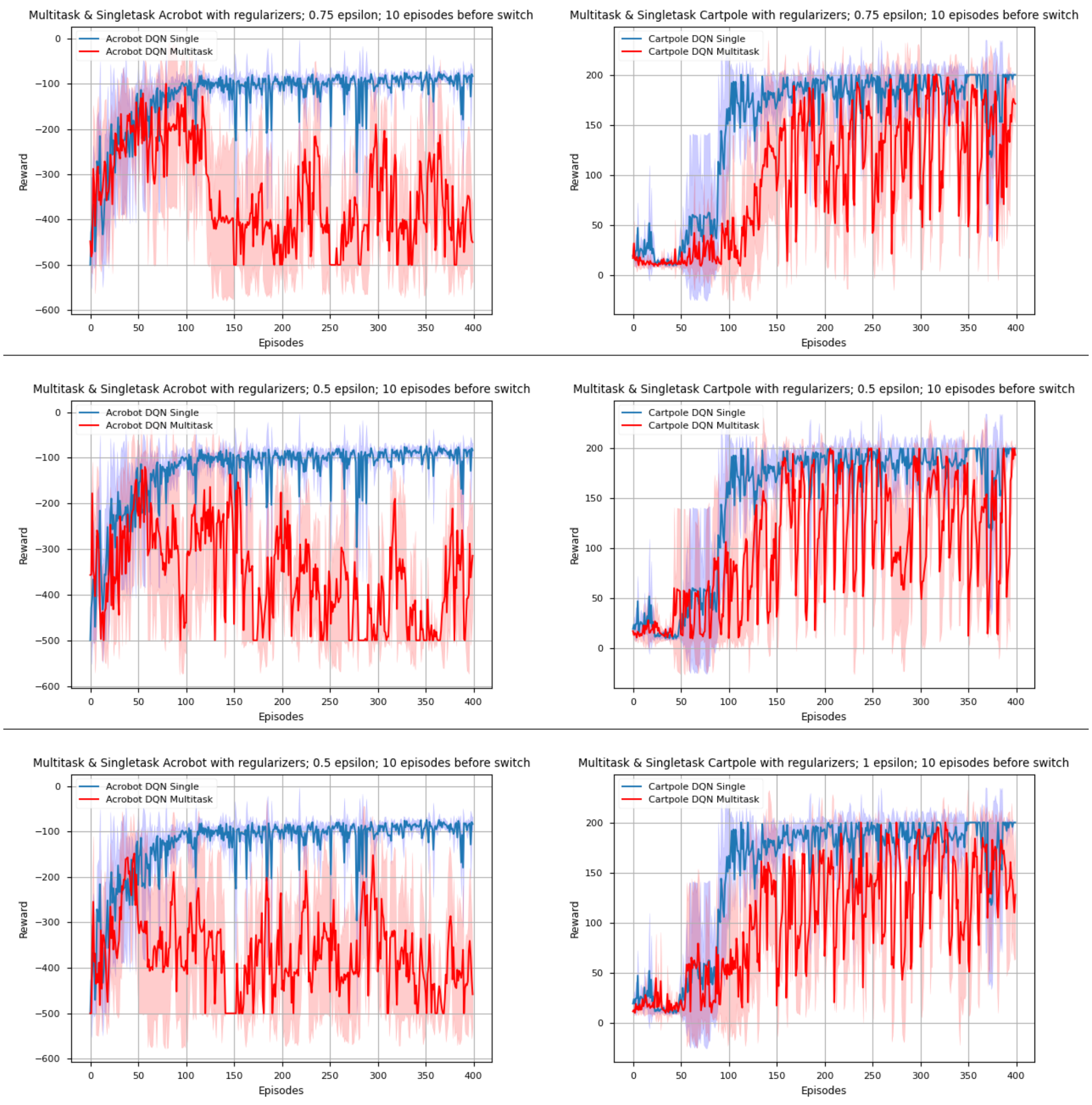


**Figure C.4: Dual-Task results with regularizers and 10 episodes in between switching episodes.**

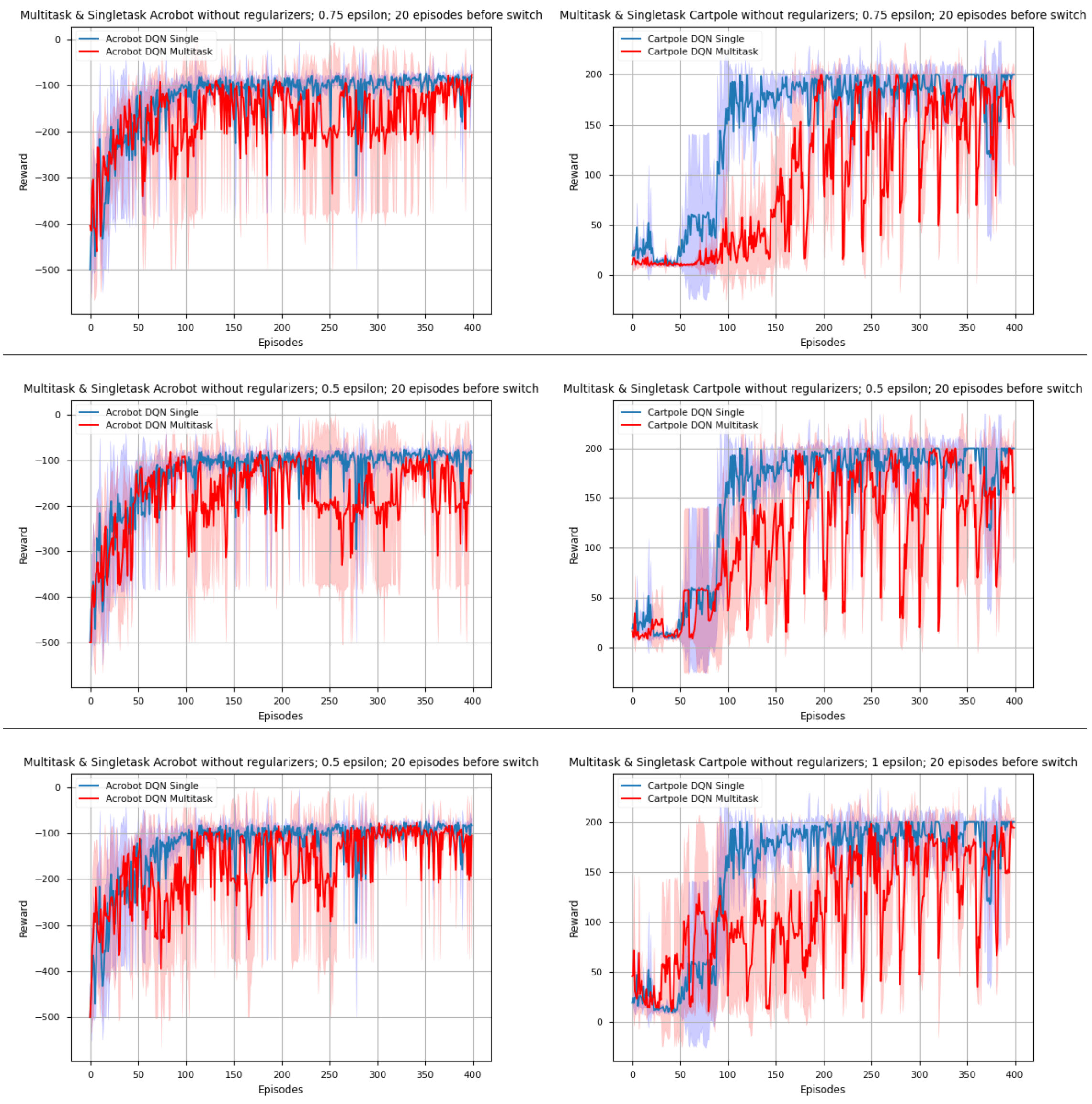


**Figure C.5: Dual-Task results without regularizers and 20 episodes in between switching episodes.**

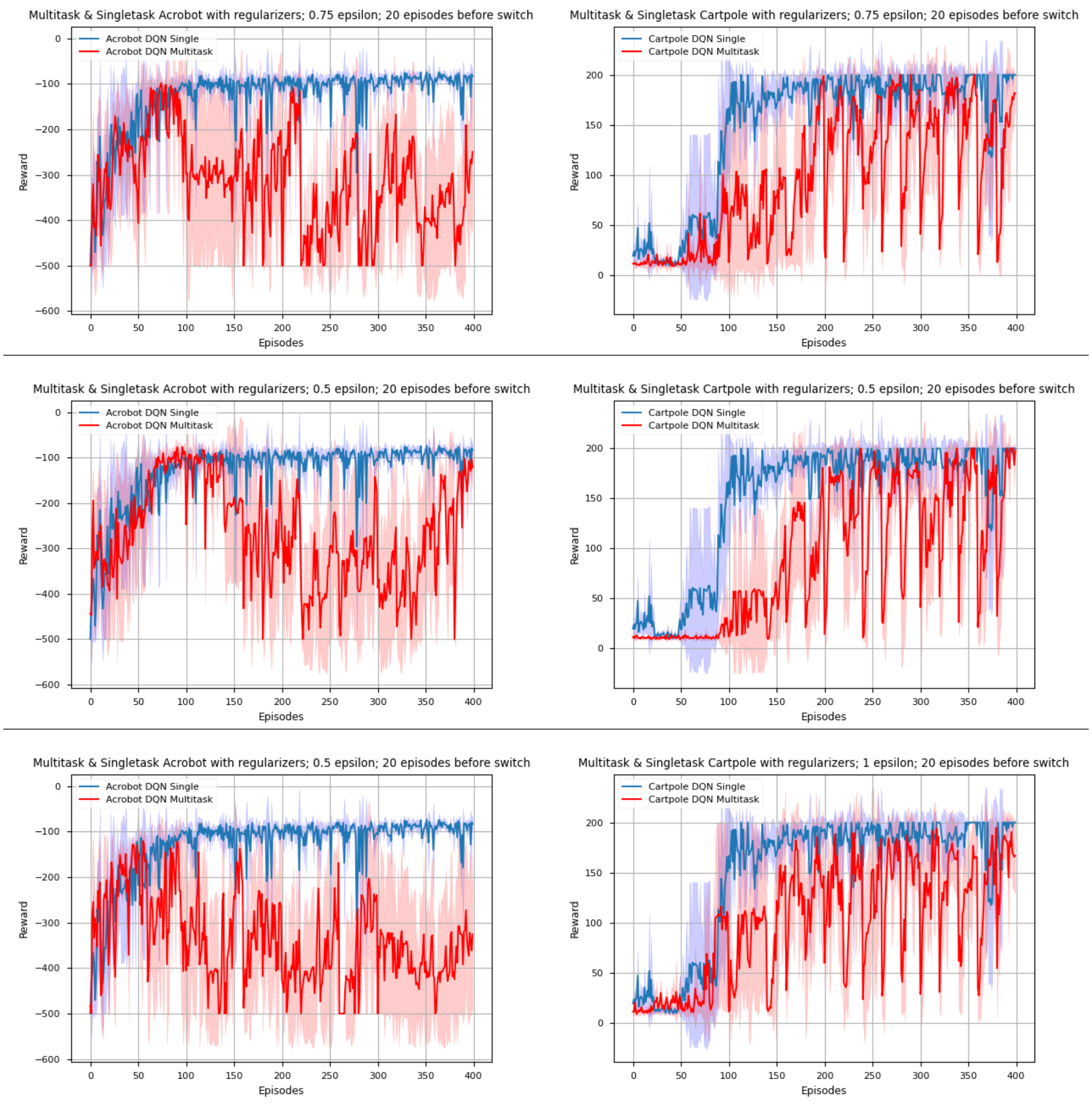


**Figure C.6: Dual-Task results with regularizers and 20 episodes in between switching episodes.**

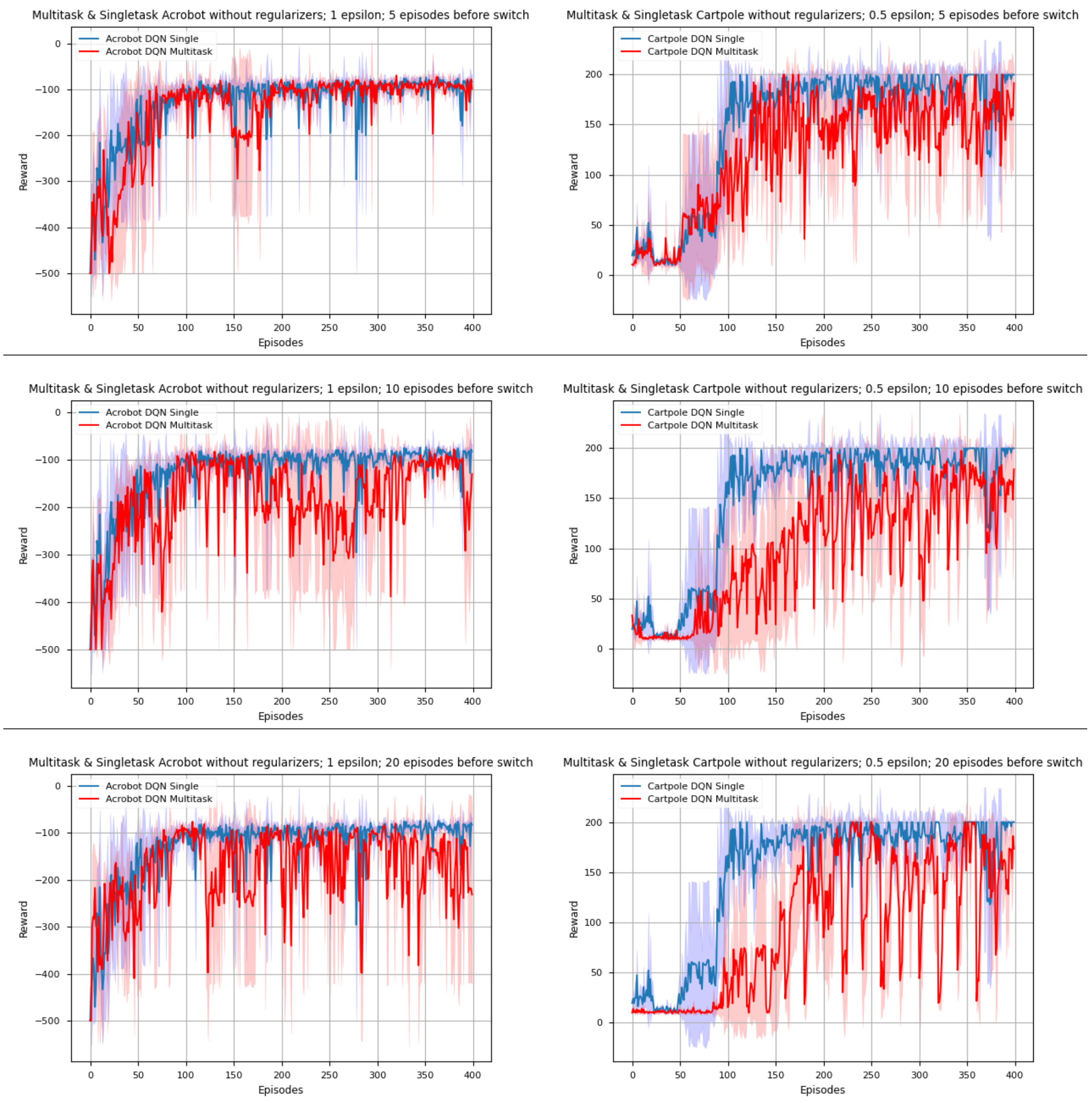


**Figure C.7: Dual-Task results without regularizers and initial epsilon values of 1 for Acrobot and 0.5 for Cartpole.**

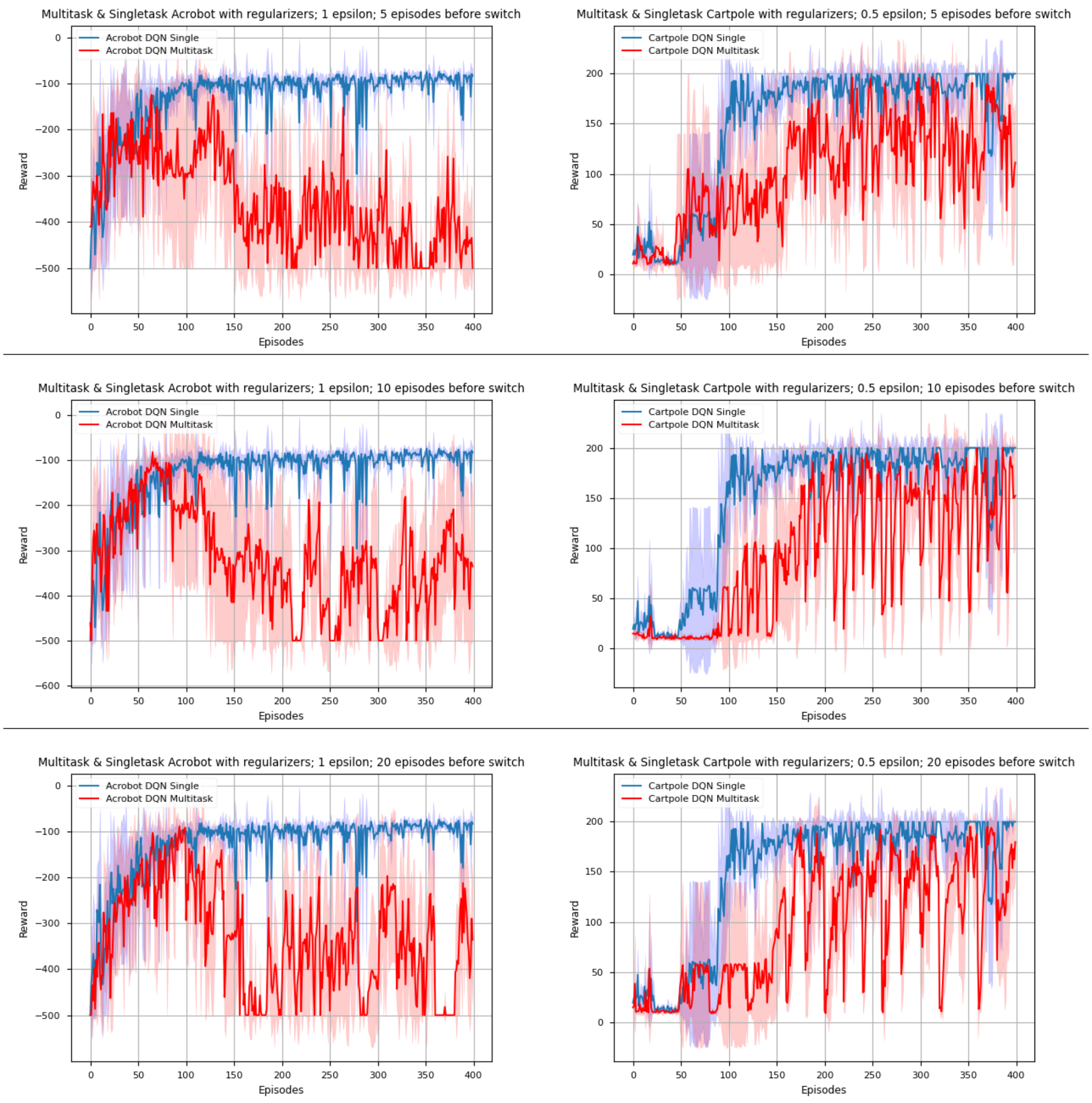


**Figure C.8: Dual-Task results with regularizers and initial epsilon values of 1 for Acrobot and 0.5 for Cartpole.**

## D Appendix - Triple-Task Model Results

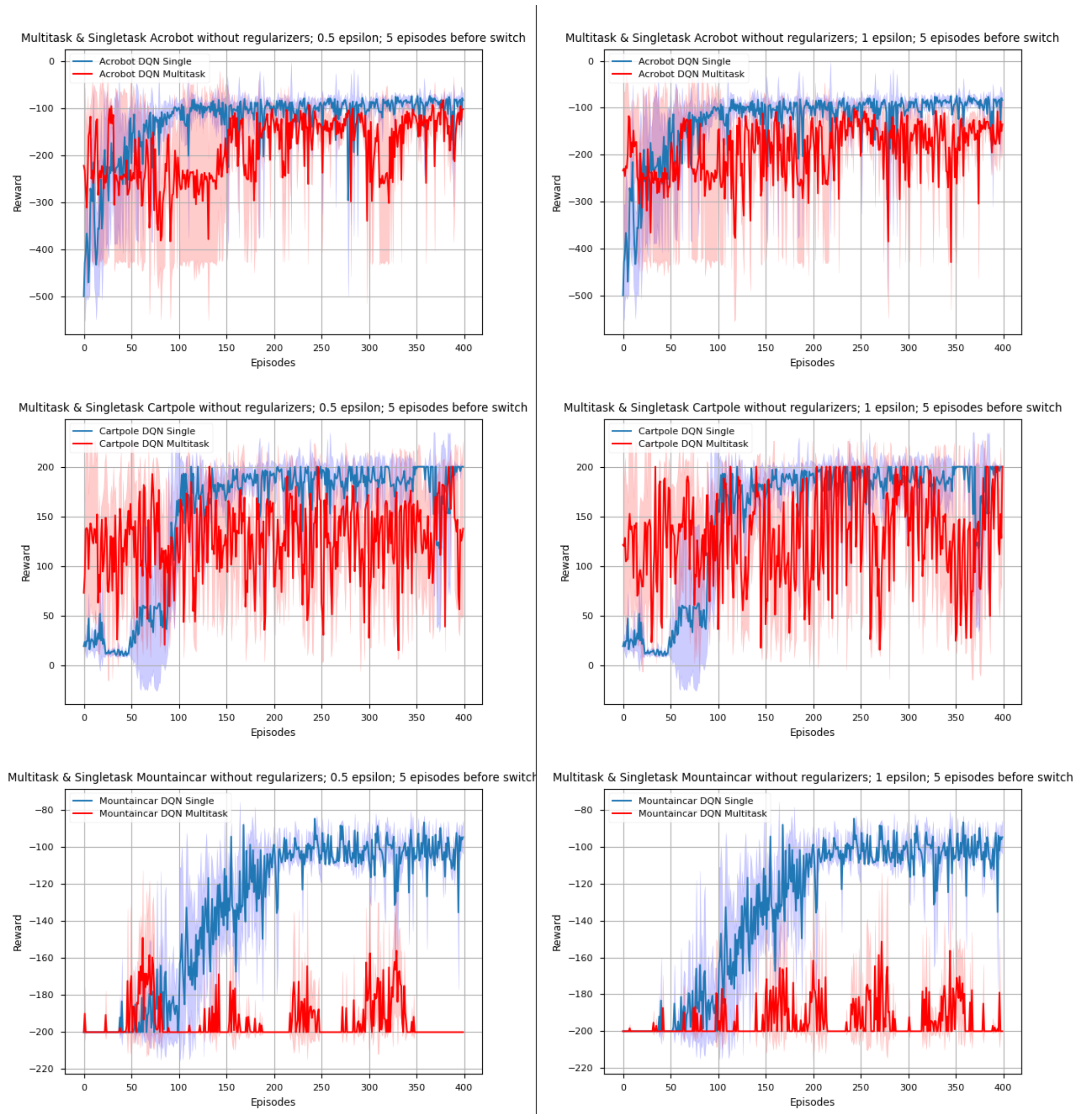


**Figure D.1: Triple-Task model results without regularization and 5 episodes in between switching. Initial epsilon values are 0.5 (left) or 1 (right) for all environments.**

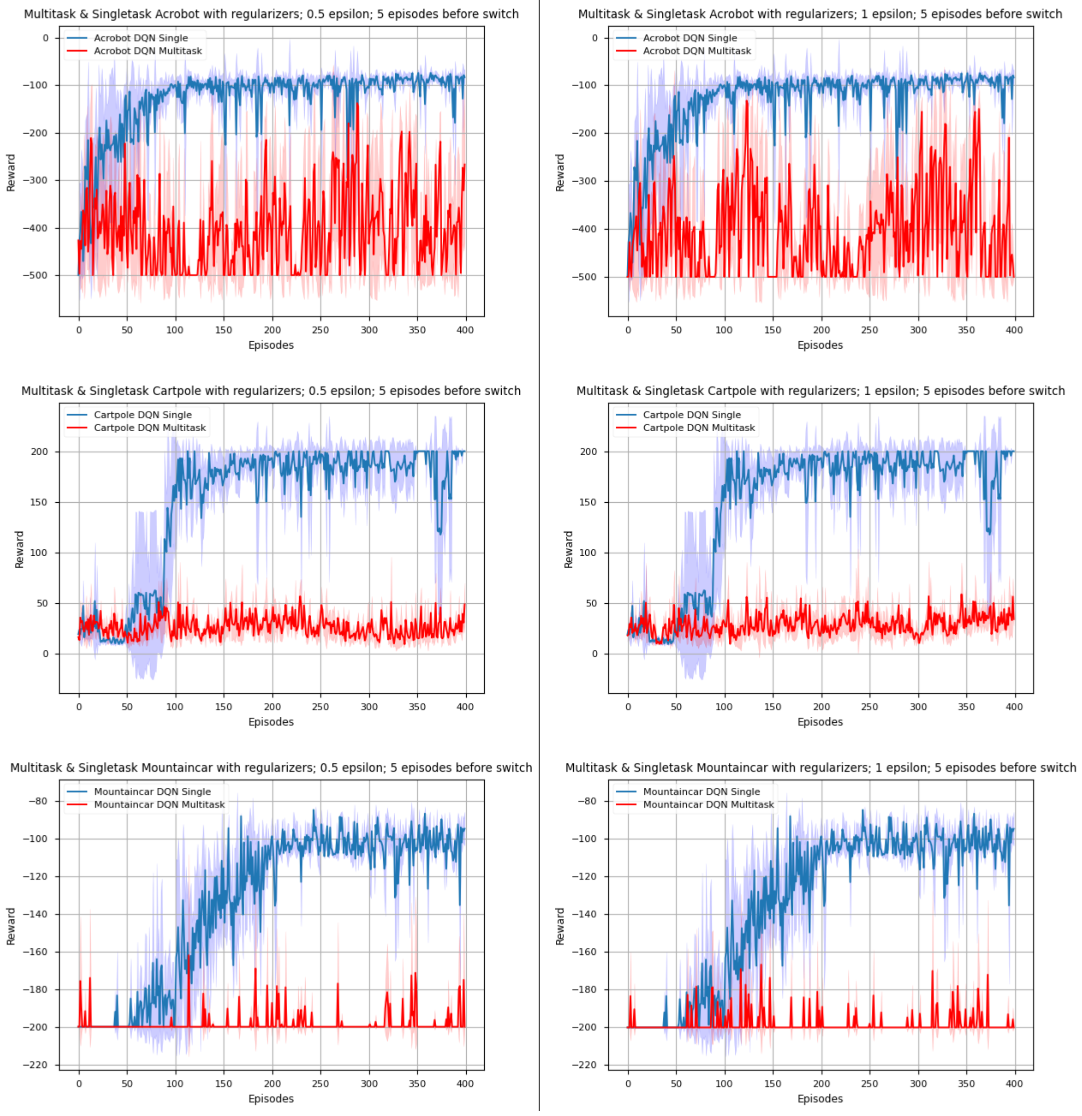


**Figure D.2: Triple-Task model results with regularization and 5 episodes in between switching. Initial epsilon values are 0.5 (left) or 1 (right) for all environments.**

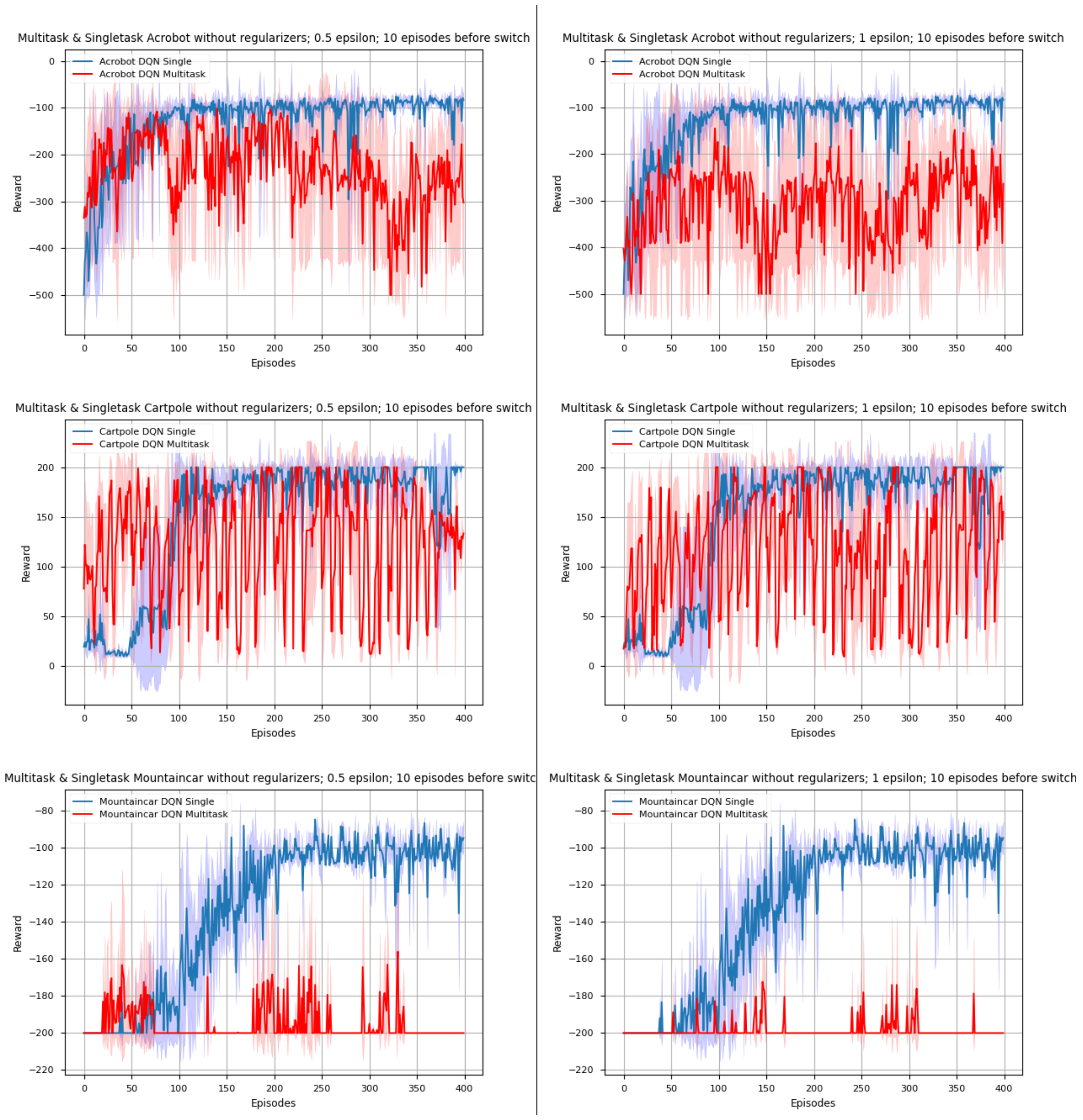


**Figure D.3: Triple-Task model results without regularization and 10 episodes in between switching. Initial epsilon values are 0.5 (left) or 1 (right) for all environments.**

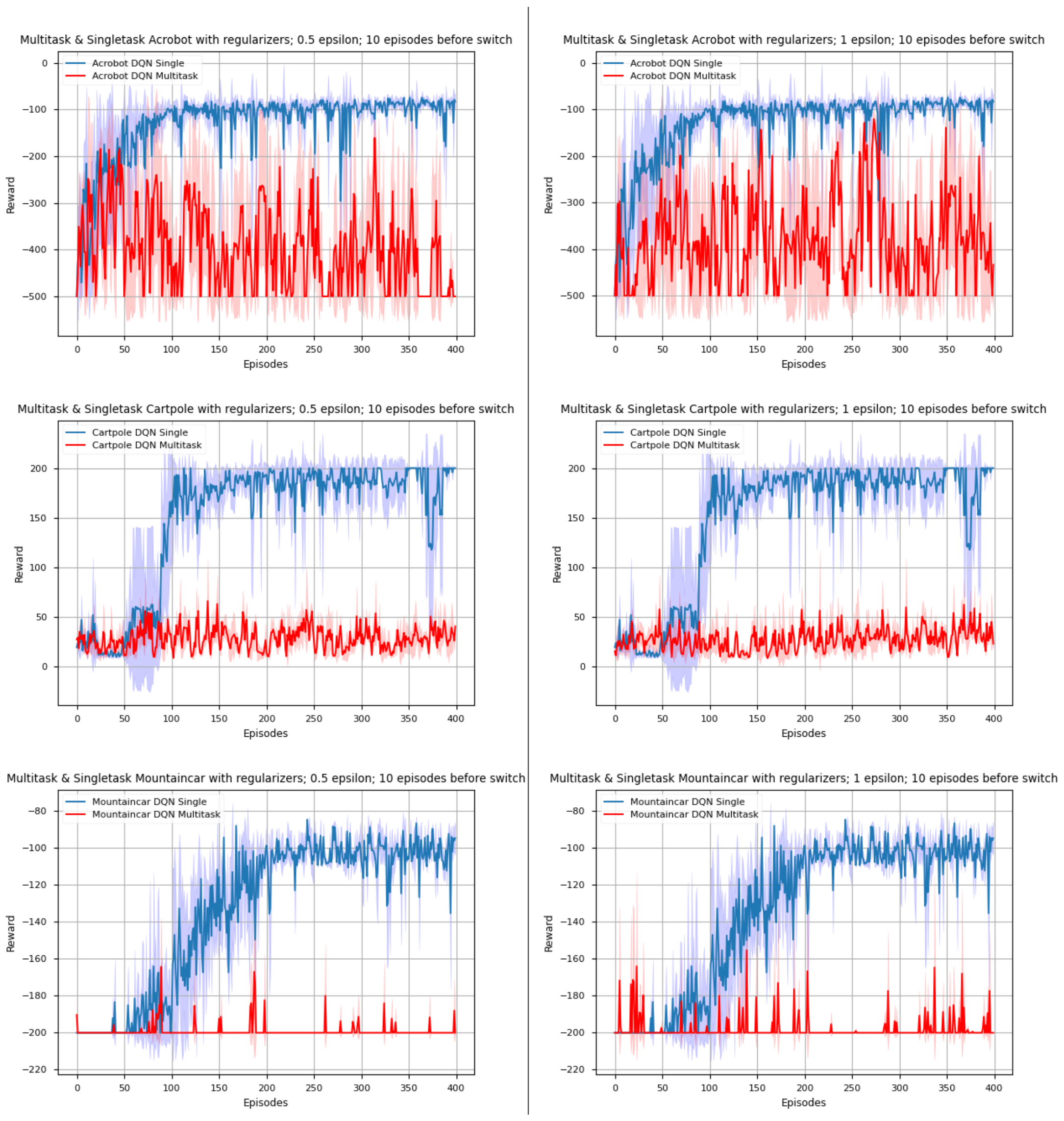


**Figure D.4: Triple-Task model results with regularization and 10 episodes in between switching. Initial epsilon values are 0.5 (left) or 1 (right) for all environments.**

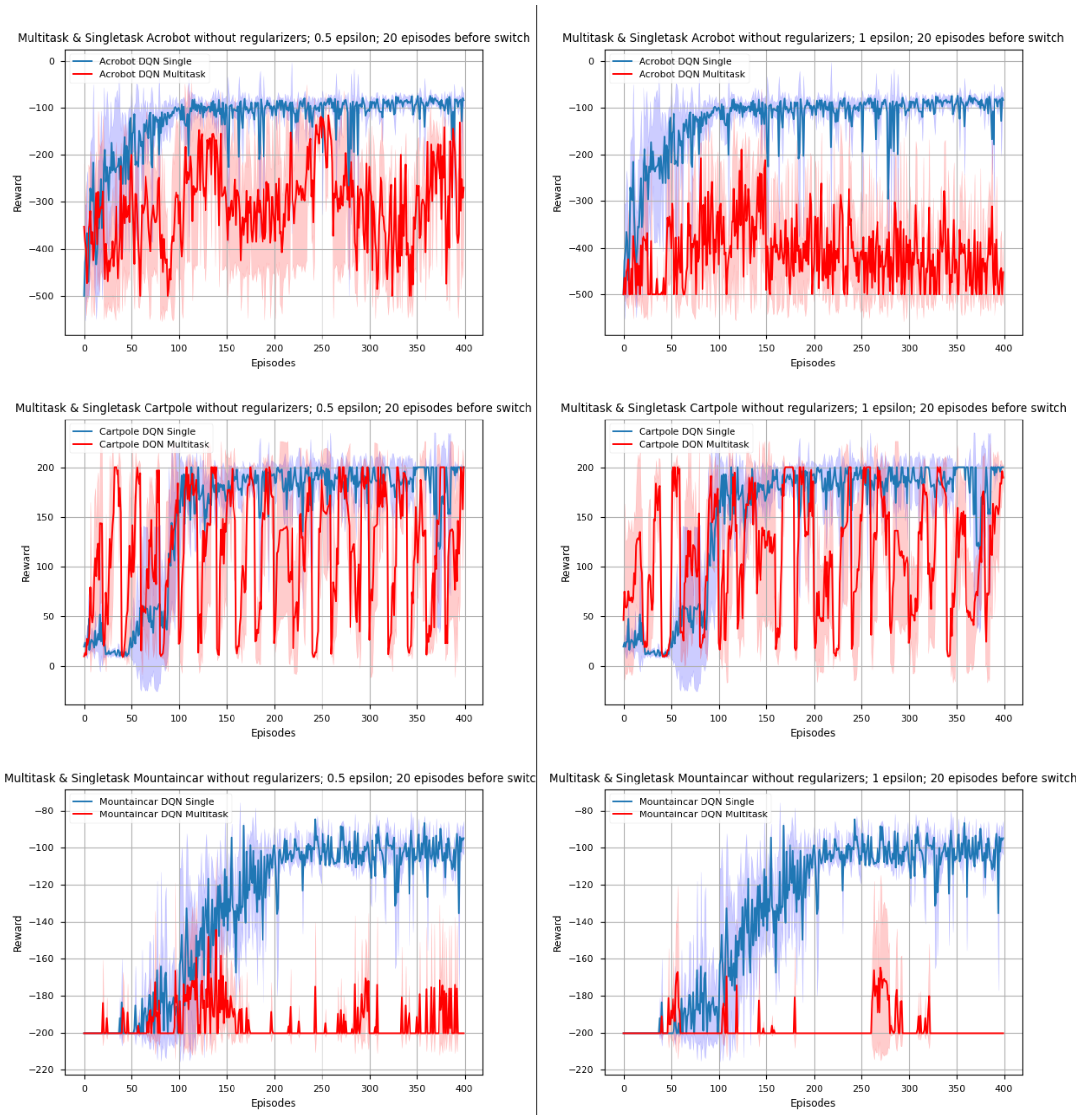


**Figure D.5: Triple-Task model results without regularization and 20 episodes in between switching. Initial epsilon values are 0.5 (left) or 1 (right) for all environments.**

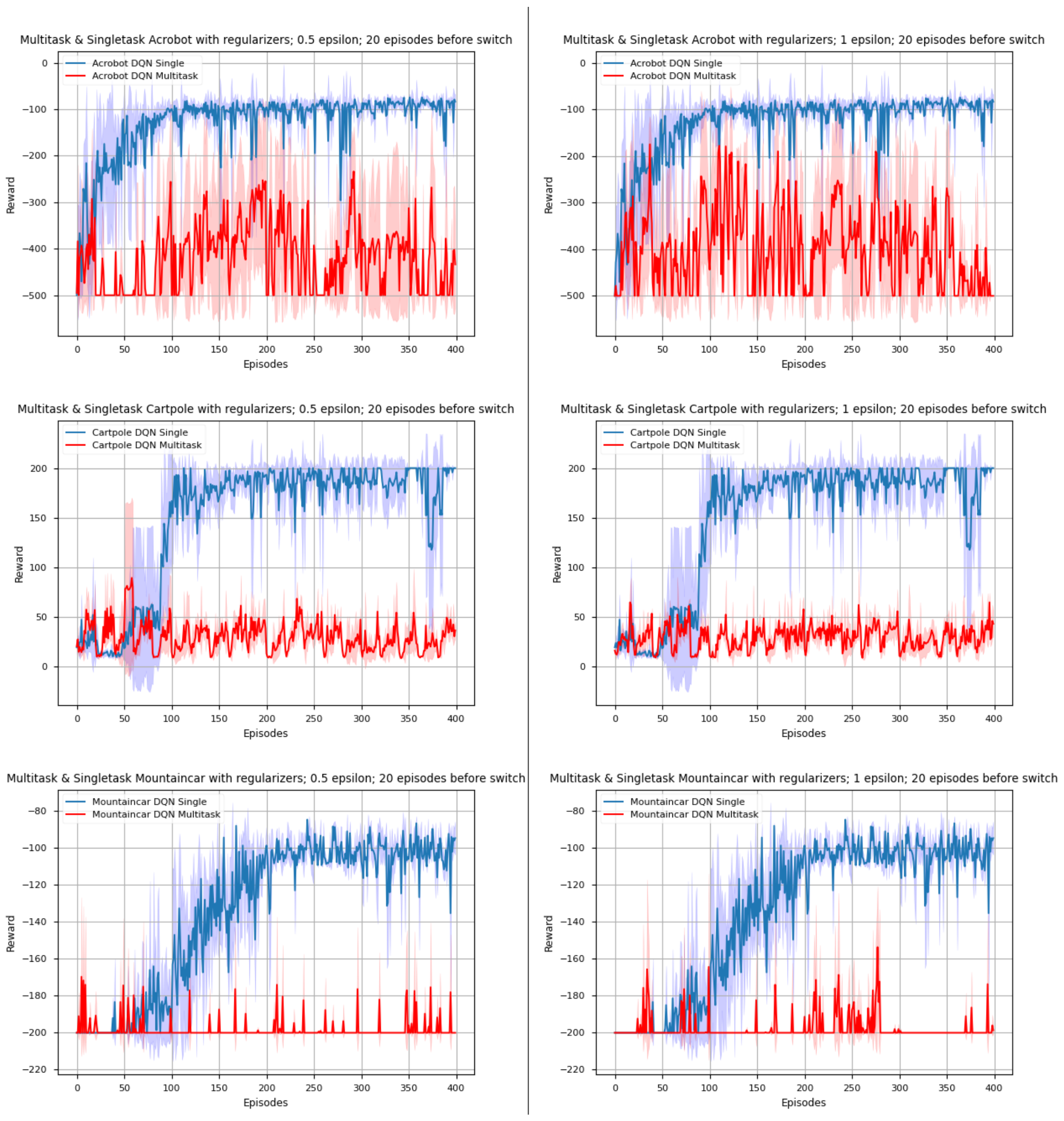


**Figure D.6: Triple-Task model results with regularization and 20 episodes in between switching. Initial epsilon values are 0.5 (left) or 1 (right) for all environments.**

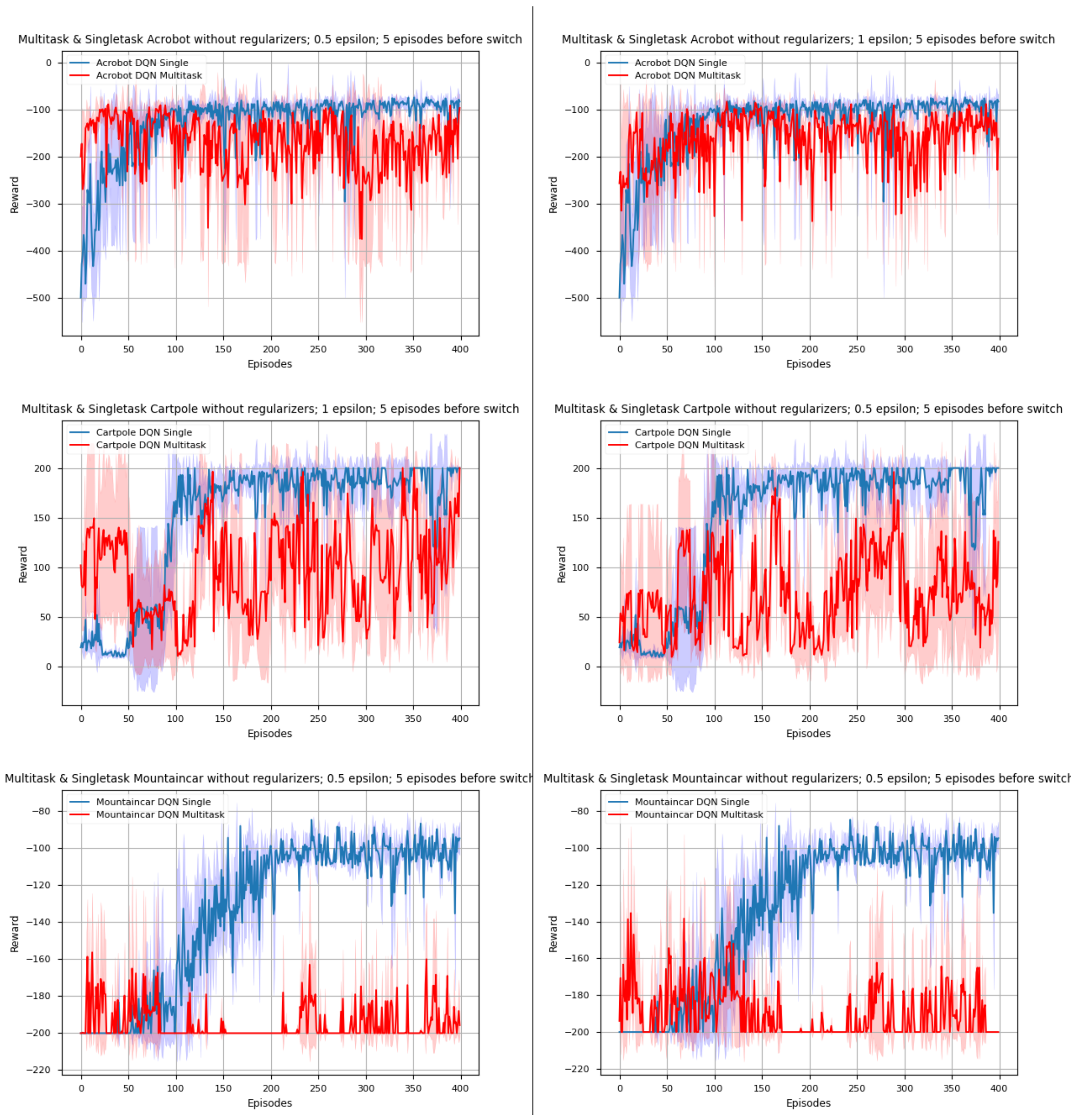


**Figure D.7: Triple-Task model results without regularization and 5 episodes in between switching. Initial epsilon values are 0.5 for AB; 1 for CP; and 0.5 for MC (left) or 1 for AB; 0.5 for CP; 0.5 for MC (right) for all environments.**

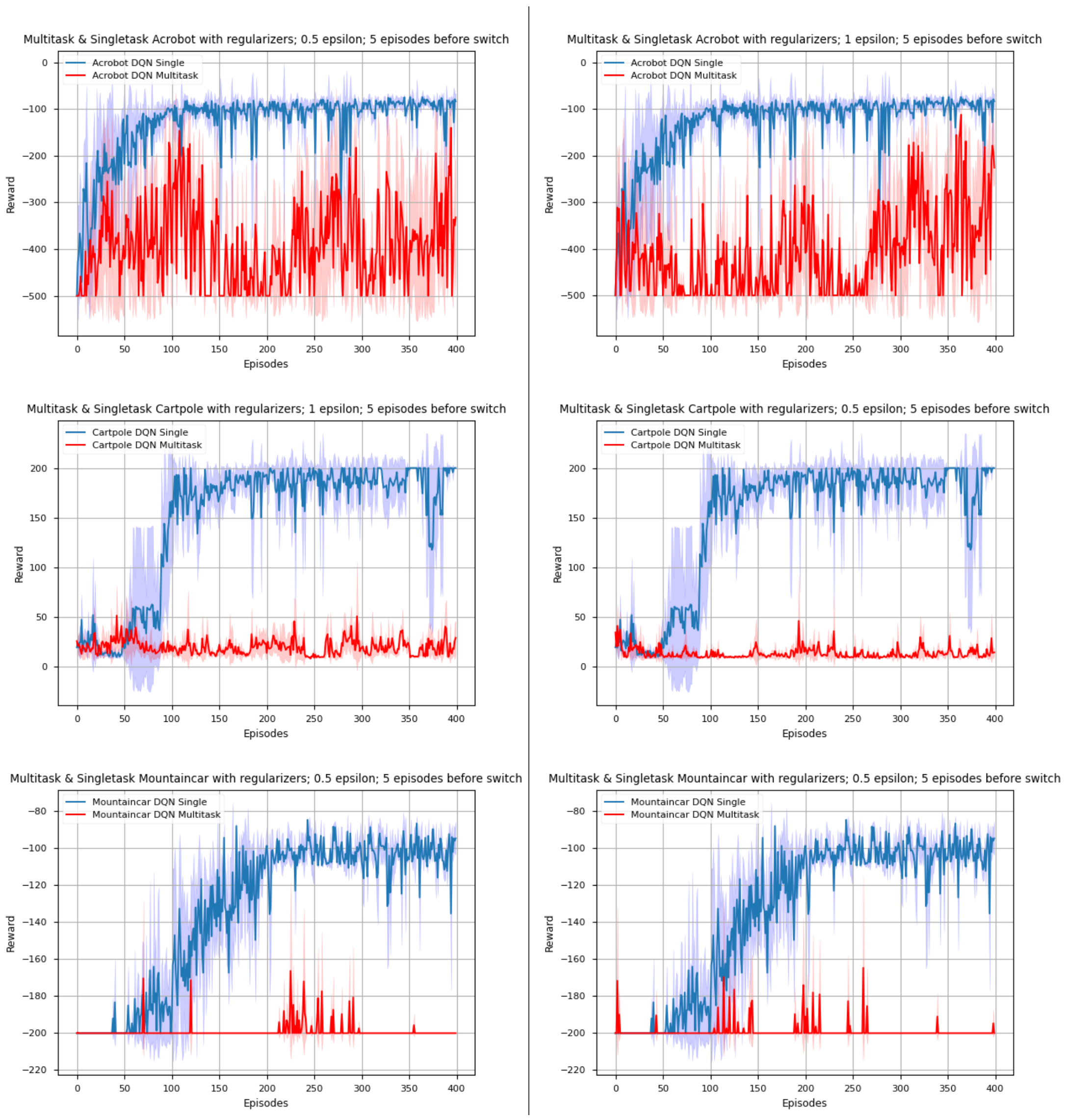


**Figure D.8: Triple-Task model results with regularization and 5 episodes in between switching. Initial epsilon values are 0.5 for AB; 1 for CP; and 0.5 for MC (left) or 1 for AB; 0.5 for CP; 0.5 for MC (right) for all environments.**

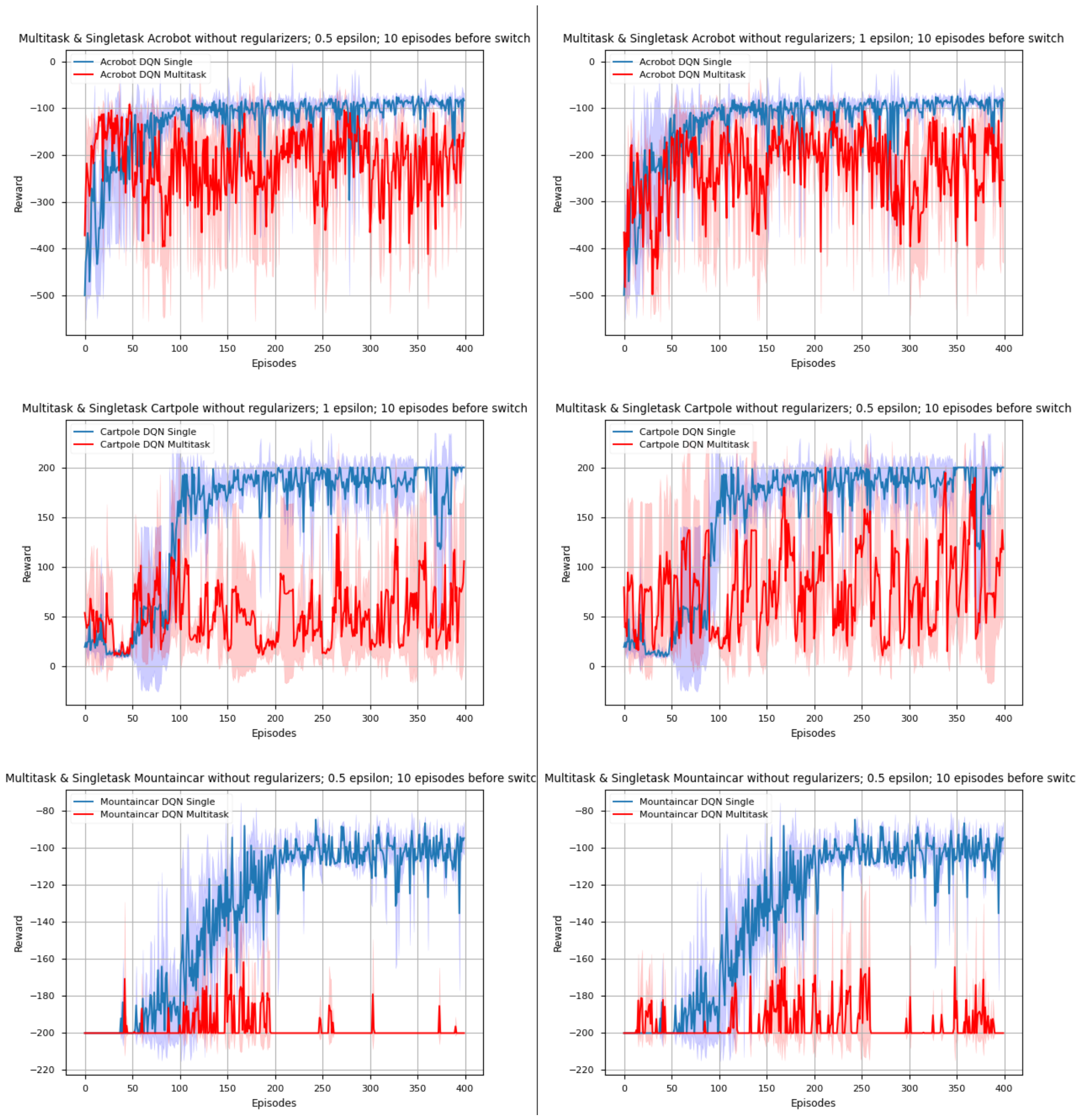


**Figure D.9: Triple-Task model results without regularization and 10 episodes in between switching. Initial epsilon values are 0.5 for AB; 1 for CP; and 0.5 for MC (left) or 1 for AB; 0.5 for CP; 0.5 for MC (right) for all environments.**

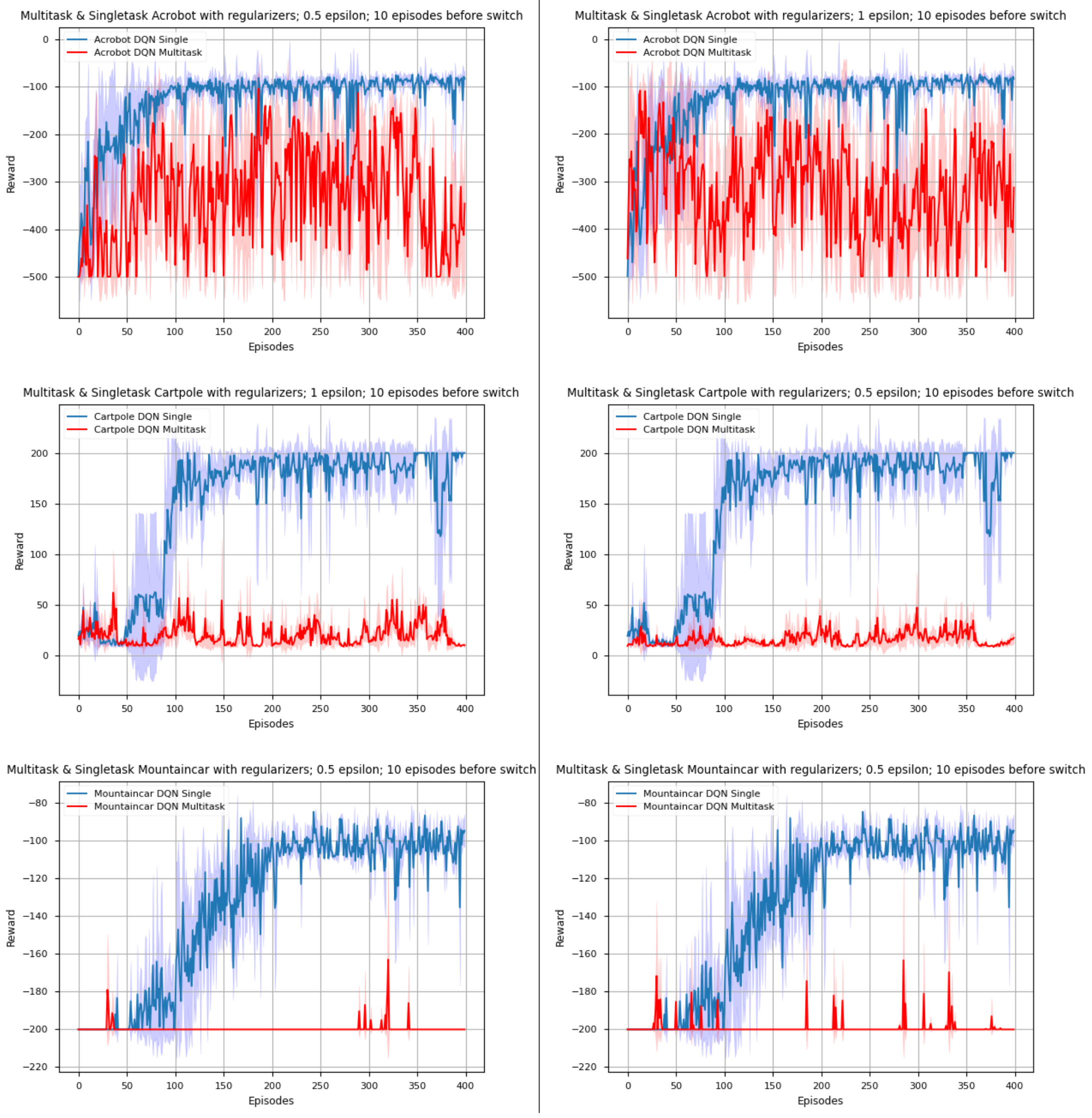


**Figure D.10: Triple-Task model results with regularization and 10 episodes in between switching. Initial epsilon values are 0.5 for AB; 1 for CP; and 0.5 for MC (left) or 1 for AB; 0.5 for CP; 0.5 for MC (right) for all environments.**

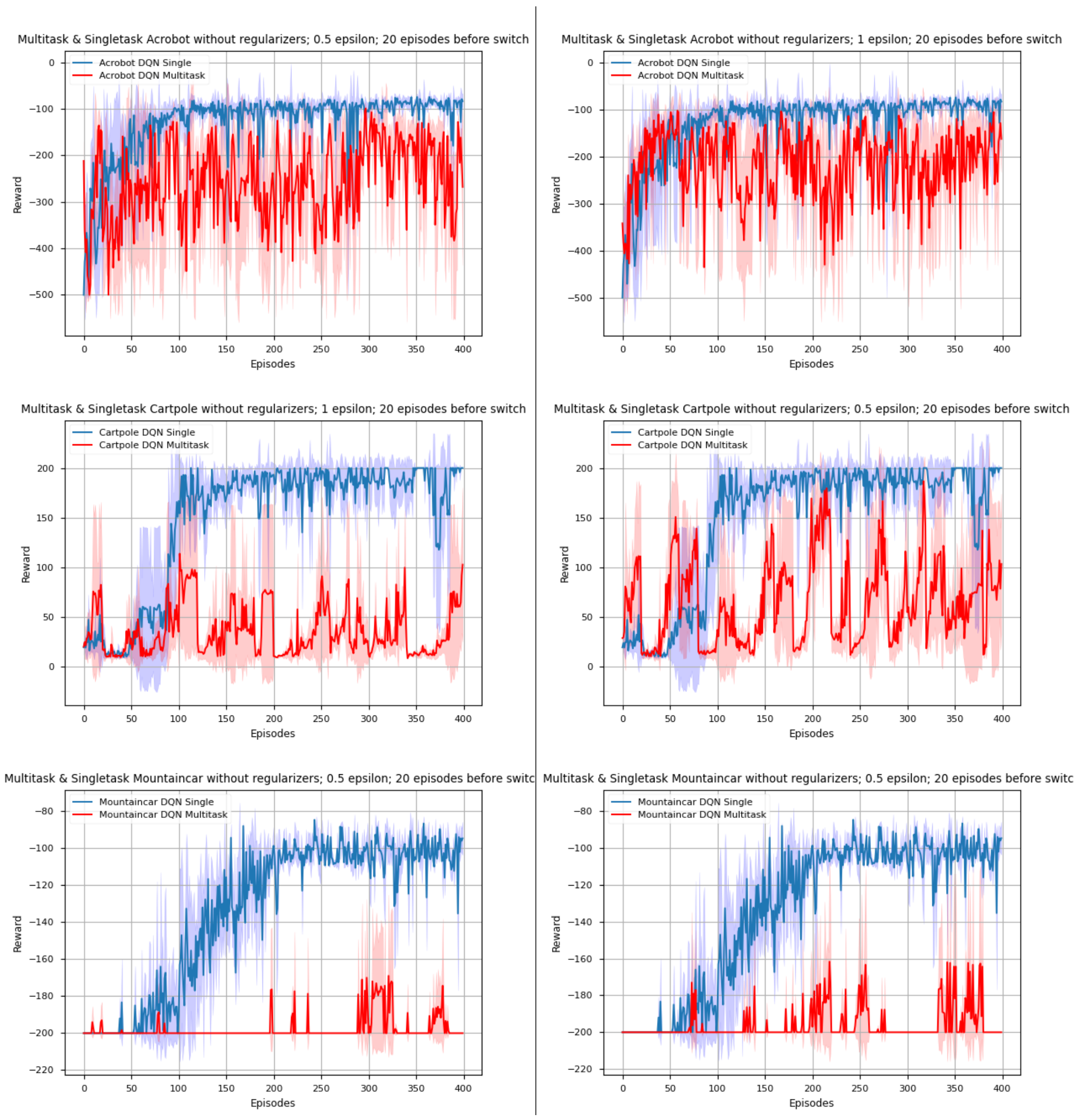


**Figure D.11: Triple-Task model results without regularization and 20 episodes in between switching. Initial epsilon values are 0.5 for AB; 1 for CP; and 0.5 for MC (left) or 1 for AB; 0.5 for CP; 0.5 for MC (right) for all environments.**

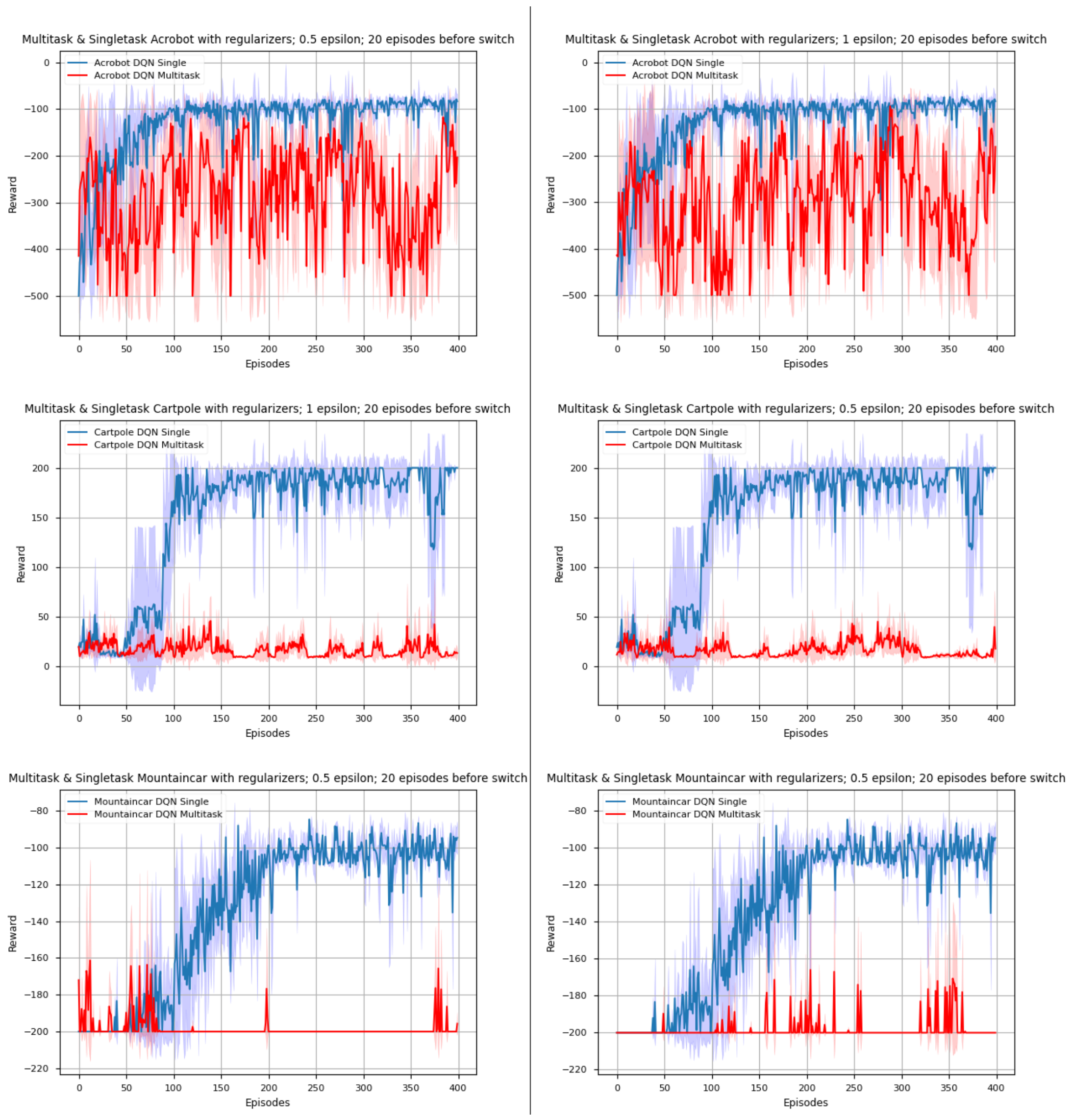


**Figure D.12: Triple-Task model results with regularization and 20 episodes in between switching. Initial epsilon values are 0.5 for AB; 1 for CP; and 0.5 for MC (left) or 1 for AB; 0.5 for CP; 0.5 for MC (right) for all environments.**

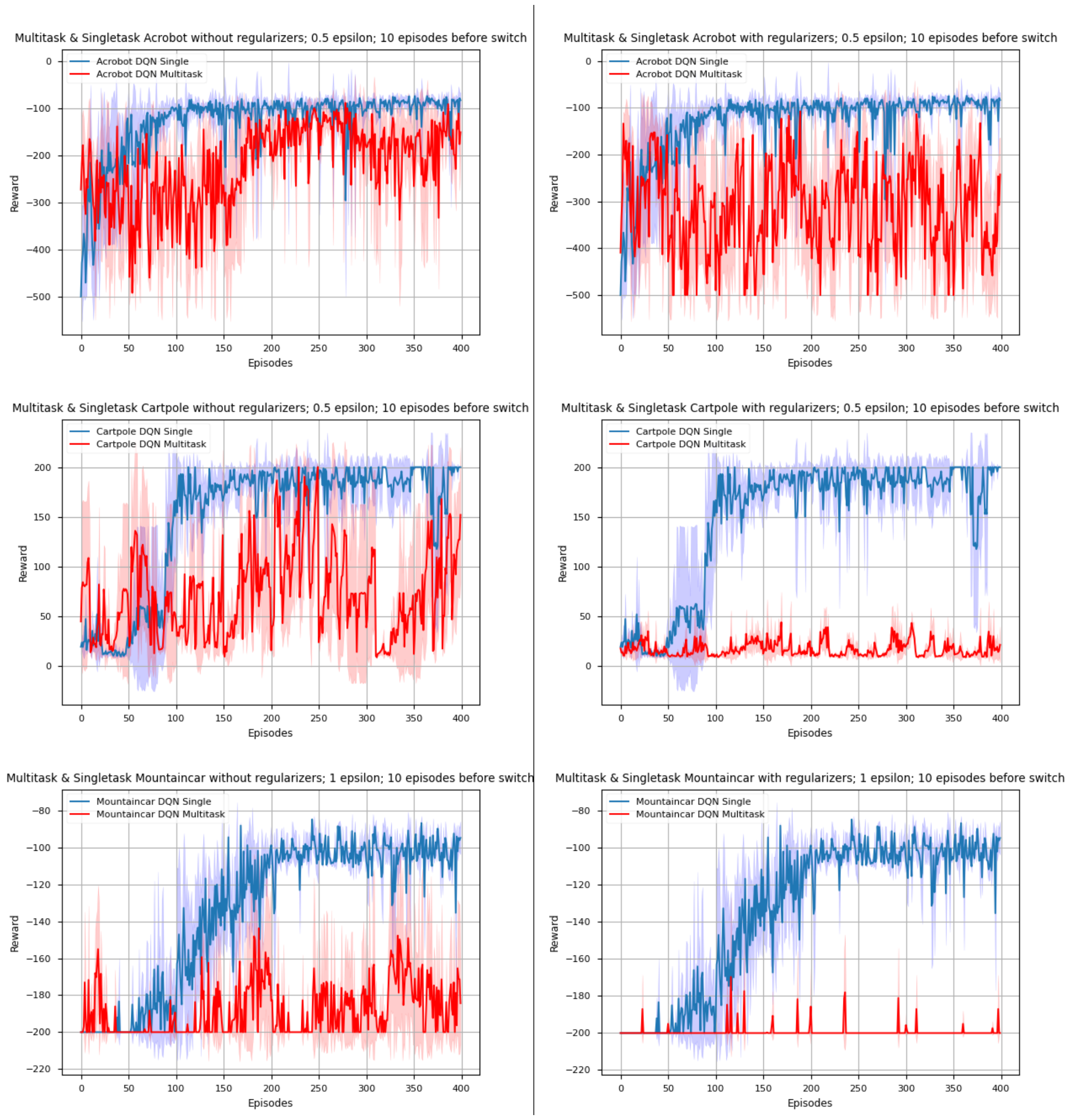


**Figure D.13: Triple-Task model results without (left) and with (right) regularization and 10 episodes in between switching. Initial epsilon values are 0.5 for AB; 0.5 for CP; and 1 for MC**

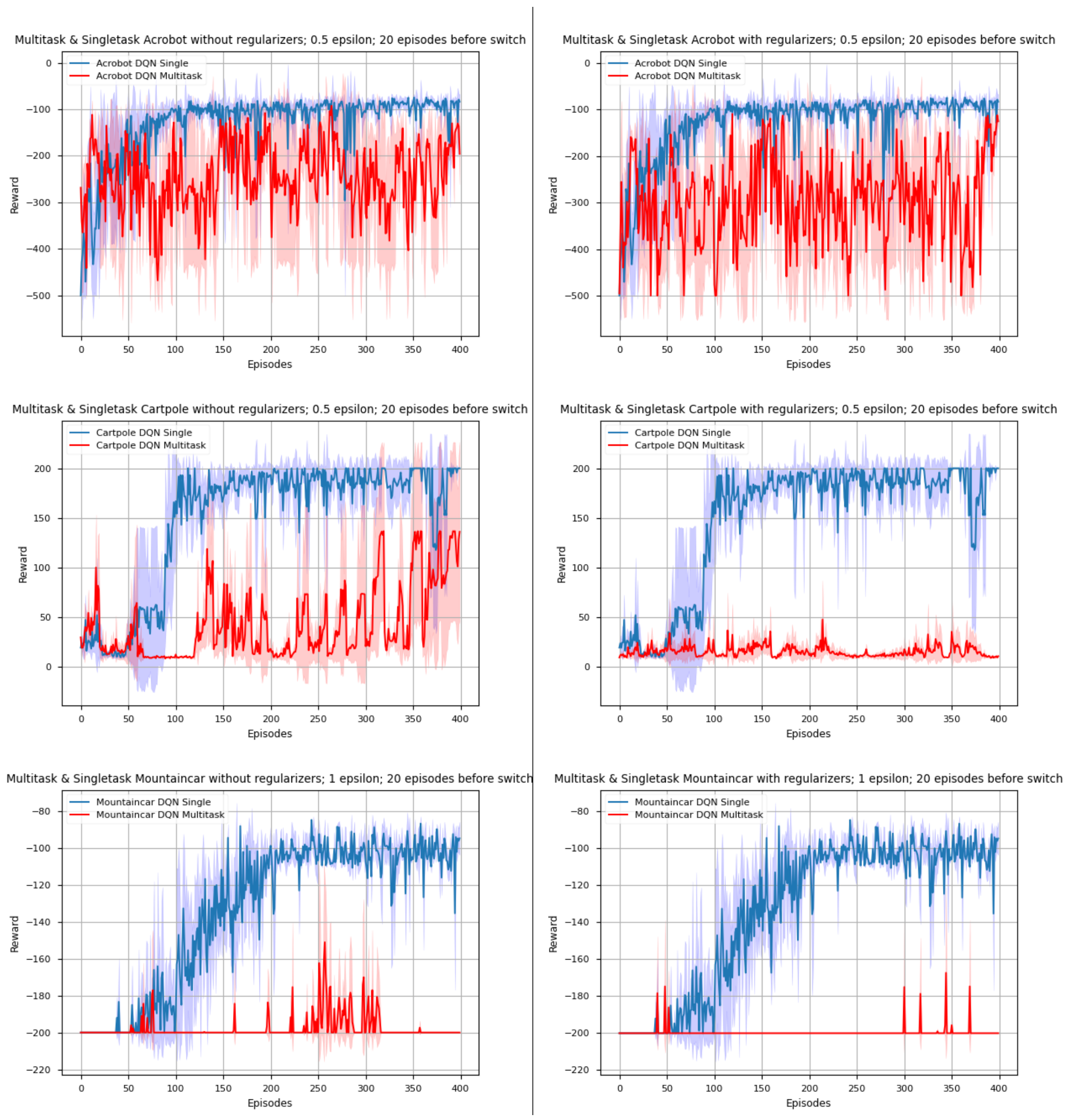


**Figure D.14: Triple-Task model results without (left) and with (right) regularization and 20 episodes in between switching. Initial epsilon values are 0.5 for AB; 0.5 for CP; and 1 for MC**