# Control of Inverted Double Pendulum using Reinforcement Learning

Fredrik Gustafsson (fregu856@stanford.edu)

December 16, 2016

#### Abstract

In this project, we apply reinforcement learning techniques to control an inverted double pendulum on a cart. We successfully learn a controller for balancing in a simulation environment using Q-learning with a linear function approximator, without any prior knowledge of the system at hand. We do however fail to learn a controller for the swing-up maneuver, which leads to a discussion on what might be needed to solve more complex control problems using reinforcement learning.

## 1 Introduction and related work

Control of dynamical systems normally requires detailed knowledge of the system at hand, which can be both very difficult and time consuming to obtain as the system complexity increases. Applying learning algorithms that can automatically learn at least near-optimal controllers without prior knowledge of the system is thus an attractive alternative approach in these situations.

Applying learning algorithms in control problems also make sense in situations where it is difficult, if not impossible, to explicitly determine which set of actions will make the system behave as desired, for example in the case of trying to make a two-legged robot walk smoothly. In cases like these one would like for the *robot* to explore different approaches and gradually learn the optimal one.

Therefore, in this project we study how to apply reinforcement learning techniques to a control problem. Specifically, we focus on the problem of controlling an inverted double pendulum on a cart. The double pendulum is chosen for this study as a proof of concept problem since it is a complex (even chaotic) dynamic system, but which also is well-studied and enables simple implementation of a system simulator. This simulator then serves as a placeholder for the actual physical system in the learning process.

Previous work where conventional control methods are used to successfully control double pendulums include [1]-[3], and the goal of this project is to replicate some of this work without utilizing any prior knowledge of the system

and instead only using reinforcement learning. Specifically, the goal is to learn controllers for basic balancing as well as for the more complex maneuver of swing-up, where the pendulum is moved from a resting position hanging vertically to an upright, balanced position. This has previously been done in [4], where a swing-up controller was learned in a low number of trails by learning a probabilistic dynamics model and explicitly incorporating the model uncertainty into planning. This method has however been shown to be too computationally demanding for practical implementation in high-dimensional problems [5], and we will thus try to solve the problem using a simpler approach.

Recent advancements in deep reinforcement learning are also starting to be applied to control problems. In [6], the DDPG algorithm is developed and solves several simulated physics tasks. It has also been successfully applied to balance an inverted double pendulum in OpenAI Gym [7], but it has yet to be applied to the swing-up problem.

## 1.1 The inverted double pendulum on a cart

The studied system consists of two joint pendulums connected to a cart that is moving on a track, see figure 1 below. If one defines the state  $\mathbf{s} \in \mathbb{R}^6$  as

$$\mathbf{s} = \begin{bmatrix} \theta_0 & \theta_1 & \theta_2 & \dot{\theta}_0 & \dot{\theta}_1 & \dot{\theta}_2 \end{bmatrix}^T, \tag{1}$$

then the dynamic equations for the system can be written

$$\dot{\mathbf{s}} = \mathbf{A}(\mathbf{s})\mathbf{s} + \mathbf{b}(\mathbf{s}) + \mathbf{c}(\mathbf{s})u, \tag{2}$$

where  $\mathbf{A} \in \mathbb{R}^{6 \times 6}, \mathbf{b} \in \mathbb{R}^6, \mathbf{c} \in \mathbb{R}^6$  and  $u \in \mathbb{R}$  is the applied force on the cart. See [8] for derivation and the complete expressions.

#### 2 Methods

In this section, we formulate the problem more precisely in terms of an MDP, we briefly describe the simulation environment and describe the implemented algorithms in

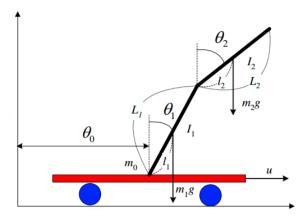


Figure 1: An inverted double pendulum on a cart [8].

### 2.1 MDP formulation

The *state* of the MDP is simply the state  $\mathbf{s} \in \mathbb{R}^6$  of the system as given by (1) above. Thus, the state space is continuous. The *actions* of the agent corresponds to choosing what force u should be applied to the cart. Depending on the specific algorithm, we will deal with both discrete (u can only take on a finite set of values) and continuous (u can take on any value in a certain interval) action spaces a. The *state transition probabilities* are unknown to the agent, but intrinsically given by (2) above.

To specify the rewards, we specify two different types of costs and then set the reward to equal the negative total cost. The first type of cost is associated with every failed episode. For swing-up, we define an episode as failed once the pendulum system moves out of bound, that is, once  $|\theta_0|$  is greater than some threshold value. For balancing, an episode is also defined as failed once the pendulum falls over, that is, once  $|\theta_1| > \frac{\pi}{2}$ . The second type of cost is associated with each state and penalizes any position of the pendulum system such that

$$heta = egin{bmatrix} heta_0 & heta_1 & heta_2 \end{bmatrix}^T 
eq egin{bmatrix} 0 & n2\pi & m2\pi \end{bmatrix}^T,$$

where  $n, m \in \mathbb{Z}$ . Finally, the discount factor was set to  $\gamma = 0.99$ .

#### 2.2 Simulation environment

The physical system is simulated using a modified version of the CartPole-v0 environment in OpenAI Gym [7]. In each time step, the agent gets to choose an action a = u which is applied to the system in its current state  $\mathbf{s}_t$ . The system is then propagated to the next time step using the dynamic equations in (2) above and numerical integration using a Runge-Kutta method. The agent now receives an observation of this new state  $\mathbf{s}_{t+1}$  of the system, as well as the immediate reward  $r_{t+1}$  associated with taking the

chosen action a in the original state  $\mathbf{s}_t$ . The current state is also rendered in each time step, see figure 2 below.

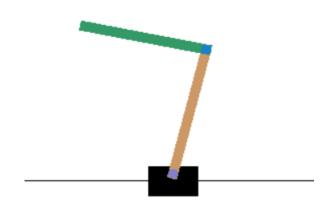


Figure 2: The simulation environment.

## 2.3 Algorithms

In this section, which is based on the lectures from David Silver's course on reinforcement learning [9], we describe two different types of algorithms. First, we describe linear Q-learning which is a value based algorithm, meaning that we learn a value function and then use an implicit policy based off of this value function. In contrast, policy based algorithms learn a policy directly and use no value function. The second algorithm we describe, Gaussian QAC, learns both a value function and an explicit policy. This type of algorithm, which can be viewed as a combination of a value based and a policy based algorithm, is called an actor-critic.

# 2.3.1 Linear Q-learning

Since the state space is continuous, we can not implement a regular Q-learning algorithm. Instead, we use linear function approximation. That is, we approximate the true action-value function  $q_{\pi}$  by a linear combination of n stateaction features,

$$q_{\pi}(\mathbf{s}, a) \approx \sum_{j=1}^{n} w_j x_j(\mathbf{s}, a) = \mathbf{x}(\mathbf{s}, a)^T \mathbf{w} = \hat{q}(\mathbf{s}, a, \mathbf{w}),$$
 (3)

where  $\mathbf{x}(\mathbf{s}, a) \in \mathbb{R}^n$  is the *feature vector*. Given features  $\mathbf{x}$ , our goal is to choose the parameter  $\mathbf{w} \in \mathbb{R}^n$  such that the mean-squared error between  $\hat{q}$  and  $q_{\pi}$  is minimized. Doing so using stochastic gradient descent yields the update rule

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha (q_{\pi}(\mathbf{s}, a) - \hat{q}(\mathbf{s}, a, \mathbf{w})) \mathbf{x}(\mathbf{s}, a),$$

where  $\alpha \in \mathbb{R}$  is the step size. Since  $q_{\pi}(\mathbf{s}, a)$  is unknown, we replace it by the Q-learning target and obtain the actual update rule for  $\mathbf{w}$ :

$$\triangle \mathbf{w} = \alpha (r' + \gamma \max_{a'} \hat{q}(\mathbf{s}', a', \mathbf{w}) - \hat{q}(\mathbf{s}, a, \mathbf{w})) \mathbf{x}(\mathbf{s}, a), \quad (4)$$

$$\mathbf{w} \leftarrow \mathbf{w} + \triangle \mathbf{w},$$
 (5)

where r' is the immediate reward associated with taking action a in state  $\mathbf{s}$ , and  $\mathbf{s}'$  is the successor state. Note that the action space A in this case is discrete, thus turning the  $\max_{a'}$  operation in (4) into a simple comparison.

If we in each time step improve the policy by acting  $\varepsilon$ -greedily with respect to  $\hat{q}$ , meaning that we choose  $a = \arg\max_{a'} \hat{q}(\mathbf{s}, a', \mathbf{w})$  with probability  $1 - \varepsilon$  and a random action in A with probability  $\varepsilon$ , and apply the above update rule, we obtain the linear Q-learning algorithm described in algorithm 1 below. Since this is an off-policy Temporal-Difference algorithm, convergence is *not* guaranteed [9].

initialize parameters  $\mathbf{w}_0$ ;

```
repeat
```

```
start new episode;

initialize state \mathbf{s}_0;

t = 0;

repeat

choose action a_t from \mathbf{s}_t by \varepsilon-greedy policy

w.r.t to \hat{q}(\mathbf{s}_t, a', \mathbf{w}_t);

take action a_t;

observe r_{t+1} and \mathbf{s}_{t+1};

a = \arg\max_{a'} \hat{q}(\mathbf{s}_{t+1}, a', \mathbf{w}_t);

\delta = r_{t+1} + \gamma \hat{q}(\mathbf{s}_{t+1}, a, \mathbf{w}_t) - \hat{q}(\mathbf{s}_t, a_t, \mathbf{w}_t);

\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha \delta \mathbf{x}(\mathbf{s}_t, a_t);

t \leftarrow t+1;

until failed episode or maximum number of time steps reached;
```

until convergence;

**Algorithm 1:** Linear Q-learning.

#### 2.3.2 Gaussian QAC

In policy based reinforcement learning, we parametrize the policy  $\pi_{\mathbf{v}}(\mathbf{s}, a) = p(a|\mathbf{s}; \mathbf{v})$  and try to find the parameters  $\mathbf{v} \in \mathbb{R}^m$  that maximize the policy objective function  $J(\mathbf{v})$ . Doing so using stochastic gradient ascent, and the policy gradient theorem, yields the update rule

$$\mathbf{v} \leftarrow \mathbf{v} + \alpha \nabla_{\mathbf{v}} (\log \pi_{\mathbf{v}}(\mathbf{s}, a)) q_{\pi_{\mathbf{v}}}(\mathbf{s}, a),$$

where  $\alpha \in \mathbb{R}$  is the step size. Since  $q_{\pi_{\mathbf{v}}}(\mathbf{s}, a)$  is unknown, we replace it by an estimated action-value function  $\hat{q}(\mathbf{s}, a, \mathbf{w}) \approx q_{\pi_{\mathbf{v}}}(\mathbf{s}, a)$  and obtain the actual update rule for  $\mathbf{v}$ :

$$\mathbf{v} \leftarrow \mathbf{v} + \alpha \nabla_{\mathbf{v}} (\log \pi_{\mathbf{v}}(\mathbf{s}, a)) \hat{q}(\mathbf{s}, a, \mathbf{w}).$$
 (6)

We thus have two set of parameters:  $\mathbf{v}$ , which is updated by the *actor* according to (6), and  $\mathbf{w}$ , which is updated by the *critic*. In Gaussian QAC, we use a linear function approximator for the action-value function, meaning that  $\hat{q}(\mathbf{s}, a, \mathbf{w})$  thus is given by (3) above.

We also choose a Gaussian policy, meaning that  $a|\mathbf{s}; \mathbf{v} \sim \mathcal{N}(\mu(\mathbf{s}), \sigma^2)$  where the mean is a linear combination of m state features,

$$\mu(\mathbf{s}) = \sum_{j=1}^{m} v_j \phi_j(\mathbf{s}) = \phi(\mathbf{s})^T \mathbf{v}.$$
 (7)

This choice yields a particularly simple expression for the score function  $\nabla_{\mathbf{v}}(\log \pi_{\mathbf{v}}(\mathbf{s}, a))$ ,

$$\nabla_{\mathbf{v}}(\log \pi_{\mathbf{v}}(\mathbf{s}, a)) = \frac{(a - \mu(\mathbf{s}))\phi(\mathbf{s})}{\sigma^2}.$$
 (8)

Since the action space A now is continuous, thus making it more involved to find  $\max_{a'} \hat{q}(\mathbf{s}', a', \mathbf{w})$ , we modify the update rule in (4)-(5) slightly,

$$\mathbf{w} \leftarrow \mathbf{w} + \beta(r' + \gamma \hat{q}(\mathbf{s}', a', \mathbf{w}) - \hat{q}(\mathbf{s}, a, \mathbf{w}))\mathbf{x}(\mathbf{s}, a), \tag{9}$$

where  $\beta \in \mathbb{R}$  is the step size and a' is the action given by the policy  $\pi_v$  in s'. Applying the update rules in (6) and (9) above in each time step, gives the Gaussian QAC algorithm described in algorithm 2 below.

```
initialize parameters \mathbf{w}_0 and \mathbf{v}_0;
```

start new episode;

```
repeat
```

```
initialize state \mathbf{s}_0;

sample a_0 \sim \mathcal{N}(\mu(\mathbf{s}_0), \sigma^2);

t = 0;

repeat
\begin{vmatrix} \text{take action } a_t; \\ \text{observe } r_{t+1} \text{ and } \mathbf{s}_{t+1}; \\ \text{sample } a_{t+1} \sim \mathcal{N}(\mu(\mathbf{s}_{t+1}), \sigma^2); \\ \mathbf{v}_{t+1} = \mathbf{v}_t + \alpha \frac{(a_t - \mu(\mathbf{s}_t))\phi(\mathbf{s}_t)}{\sigma^2} \hat{q}(\mathbf{s}_t, a_t, \mathbf{w}_t); \\ \delta = r_{t+1} + \gamma \hat{q}(\mathbf{s}_{t+1}, a_{t+1}, \mathbf{w}_t) - \hat{q}(\mathbf{s}_t, a_t, \mathbf{w}_t); \\ \mathbf{w}_{t+1} = \mathbf{w}_t + \beta \delta \mathbf{x}(\mathbf{s}_t, a_t); \\ t \leftarrow t + 1; \end{vmatrix}
```

until convergence;

steps reached;

Algorithm 2: Gaussian QAC.

until failed episode or maximum number of time

# 3 Experiments and results

In this section, we describe the performed experiments in detail, present the achieved results and discuss the performance of the implemented algorithms.

#### 3.1 Balancing

In each new episode, the state was initialized by setting

$$\mathbf{s}_0 = \begin{bmatrix} 0 & \theta_{1_0} & \theta_{2_0} & 0 & 0 & 0 \end{bmatrix}^T,$$

where  $\theta_{1_0}, \theta_{2_0} \sim U[-\lambda, \lambda]$ . During training,  $\lambda = 0.1$ . A maximum limit to the number of time steps in a single episode was set to 300. If this limit was reached, the episode was defined as successful.

## 3.1.1 Linear Q-learning

The following results were obtained with exploration probability  $\varepsilon = 0.01$ , step size  $\alpha = 0.0001$ , action space  $A = \{0, \pm 5, \pm 10, \pm 20, \pm 40\}$  and the parameters **w** initialized to the zero vector. The state-action features used to approximate the action-value function were

$$\mathbf{x}(\mathbf{s},a) = \begin{bmatrix} f_1 & \dots & f_{18} \end{bmatrix}^T, \tag{10}$$

where the features  $f_1$  to  $f_{18}$  are given in table 1 below. In table 1,  $normalAngle(\phi)$  gives the difference between  $\phi$  and its closest multiple of  $2\pi$ .

Table 1: Features for linear Q-learning.

$f_2 = \theta_0$
$f_4 = normalAngle(\theta_1)$
$f_6 = normalAngle(\theta_2)$
$f_8=\dot{ heta_0}$
$f_{10}=\dot{ heta_1}$
$f_{12}=\dot{ heta_2}$
$f_{14} = a\dot{\theta_0}$
$f_{16} = a\dot{\theta}_1$
$f_{18} = a\dot{\theta_2}$

With parameters as above, the algorithm converged in 70% of training attempts. A typical learning curve is given in figure 3 below.

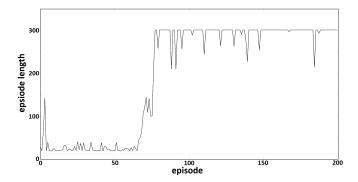


Figure 3: Typical learning curve for linear Q-learning.

To test the performance and robustness of the learned controllers, training was run until convergence ten times. Then, for each learned controller, 100 episodes was run for each  $\lambda \in \{0.1, 0.2, 0.3, 0.4\}$ . The result of this test is given in table 2 below. We see in the table that the controllers are quite robust and copes well with more challenging initialization positions for the pendulums. We do however also clearly see the limitations in their performance in the case of  $\lambda = 0.4$ .

Table 2: Performance test for linear Q-learning.

λ	% of successful episodes
0.1	94
0.2	92
0.3	78
0.4	49

A notable observation is that the learned controllers always seem to take actions  $a=\pm 40$ , that is, the actions of largest magnitude in the action space. Thus, we effectively obtain what is known as "bang-bang" controllers. This works fine in simulation, but might not be completely desirable in the case of actual implementation on a physical system. In that case, you would normally want a smoother controller to put less stress on the mechanical components. The results do however look quite promising overall. Because of the relatively quick convergence, as seen in figure 3, implementation on a physical system actually seems feasible.

A video showing the pendulum system's natural behaviour when the action a=0, excerpts from the training process and an example of a learned controller can be found at: https://goo.gl/yLZRTO.

# 3.1.2 Gaussian QAC

Despite our best efforts, we were never able find a combination of features and hyperparameters such that the algorithm converged in a reasonable number of episodes. We never observed a single successful episode in the training process either. We ran training attempts with various combinations of  $\alpha, \beta \in [10^{-8}, 10^{-1}], \sigma^2 \in [0.001, 10]$  and initializations of  $\mathbf{w}$  and  $\mathbf{v}$ . Originally, the features used were

$$\mathbf{x}(\mathbf{s}, a) = \begin{bmatrix} f_1 & \dots & f_{18} \end{bmatrix}^T,$$
$$\mu(\mathbf{s}) = \begin{bmatrix} f_1 & \dots & f_{12} \end{bmatrix}^T,$$

but a large set of additional features was also explored. For example, we tried adding different combinations of indicator variables with respect to the signs of  $\theta_0$ ,  $\theta_1$  and  $\theta_2$ . A typical learning curve is given in figure 4 below. In this partcular case, something clearly happened between episode 30000 and 40000, and it is possible that

convergence could have been reached using an even smaller step size. However, with the limited time available to the project, Gaussian QAC was simply not a usable algorithm for this problem.

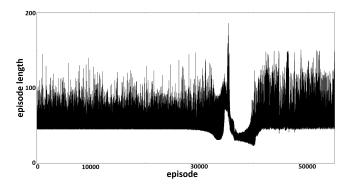


Figure 4: Typical learning curve for Gaussian QAC.

Thus, linear Q-learning clearly outperformed Gaussian QAC. We believe the main reason for this is that the latter is more sensitive to the choice of features, since two sets of parameters has to been learned. Also, the learned controllers' behaviour for linear Q-learning seems to suggest that the taken actions should be fairly large in magnitude, even for small deviations from an upright position. To achieve this using our Gaussian policy, bias terms has to be introduced. It is however not obvious how these should be designed.

## 3.2 Swing-up

In each new episode, the state was initialized by setting

$$\mathbf{s}_0 = \begin{bmatrix} 0 & \frac{\pi}{2} & \frac{\pi}{2} & 0 & 0 & 0 \end{bmatrix}^T.$$

A maximum limit to the number of time steps in a single episode was set to 1000. Since linear Q-learning clearly was the better performing algorithm for balancing, we did not consider Gaussian QAC for this problem.

Despite our best efforts however, we were never able to neither reach convergence in a reasonable number of episodes, nor observe a single episode where the control task was even close to being completed. Training attempts were run with various combinations of parameters  $\varepsilon$ ,  $\alpha$ , A and  $\mathbf{w}_0$ , and a large set of different features. To find better features, they were also evaluated by their performance on the balancing problem. We did however find this problem of designing features to be quite unintuitive. After more careful consideration, the originally used features in (10) above did not exactly seem optimal, but we still found it very difficult to beat their performance using features that intuiviely made more sense. A typical learning curve is given in figure 5 below, where "average reward" is total episode reward divided by episode length.

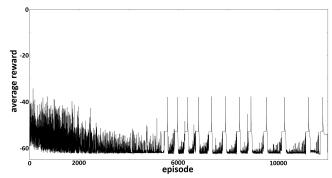


Figure 5: Typical learning curve for swing-up.

The reason why we were not able to obtain any successful results for this problem is likely because swing-up control inherently is a more difficult problem than balancing. In fact, it has been shown that this problem can not be solved using a linear controller [4]. Thus, some nonlinearities has to be introduced to the controller. Since we only use linear function approximators, this has to be done manually by carefully designing cleaver features.

## 4 Future work

If time was given to continue the work of this project, we would delve into the deep learning based techniques mentioned in the section on related work above. The results of this project seem to suggest that algorithms using only linear function approximators can be successfully applied only to relatively simple control problems. At least if you do not want to need expert-level knowledge of the specific studied system. Utilizing deep learning methods thus makes a lot of sense, as it would remove the need of hand-crafting features for any nonlinear relationship we wish to represent. As a first step, it would be interesting to study whether or not the DDPG algorithm described in [6] can be used to solve the problem of swing-up control.

#### 5 Conclusion

We studied whether relatively simple reinforcement learning algorithms can be used to control an unknown and complex dynamical system. We found it to be true only for relatively simple control problems, as we successfully learned a controller for balancing using linear Q-learning, but failed to do so for swing-up. Since we only used linear function approximators, the more challenging control problems seem to require cleverly designed features. However, choosing features is difficult and not always intuitive. Linear Q-learning clearly outperformed Gaussian QAC, likely because the latter is more sensitive to the choice of features. Future work would study the application of deep learning techniques to reinforcement learning and control.

# References

- [1] Tobias Gluck, Andreas Eder and Andreas Kugi. Swingup control of a triple pendulum on a cart with experimental validation. Automatica, volume 49, issue 3, pages 801 - 808, 2013.
- [2] Pathompong Jaiwat and Toshiyuki Ohtsuka. Real-Time Swing-up of Double Inverted Pendulum by Nonlinear Model Predictive Control. 5th International Symposium on Advanced Control of Industrial Processes, 2014.
- [3] Benjamin Jahn. Application of feedforward control design to a multi-link pendulum. Ilmenau University of Technology, Master's Thesis, 2013.
- [4] M. P. Deisenroth and C. E. Rasmussen. PILCO: A Model-Based and Data-Efficient Approach to Policy Search. International Conference on Machine Learning (ICML), 2011.
- [5] Niklas Wahlström et al. From Pixels to Torques: Policy Learning with Deep Dynamical Models. International Conference on Machine Learning (ICML), 2015.
- [6] Timothy P. Lillicrap, Jonathan J. Hunt, et al. Continuous Control With Deep Reinforcement Learning. International Conference on Learning Representations, 2016.
- [7] OpenAI Gym. https://gym.openai.com/
- [8] Alexander Bogdanov. Optimal Control of a Double Inverted Pendulum on a Cart. Technical Report CSE-04-006, 2004.
- [9] RL course by David Silver. http://www0.cs.ucl.ac.uk/staff/d.silver/web/ Teaching.html