# Machine Learning Methods for Control of Chaotic Systems

Math 516

Jadon Duby, Cole Runions-Kahler, Nathan Enerson, Steven Susanto

University of Calgary

Wednesday, March 22, 2022

## Table of Contents

## Outline

## Defining the System

The system is as follows. Two pendulums with masses $m_1$ and $m_2$ at there respective endpoints are joined together, with the mass m1 being connected via a mass-less rigid rod of length $L_2$ with the first mass being by another rigid rod of length $L_1$ to a cart which is allowed to move horizontally in either direction along a track.

Each mass-rod component is allowed to freely rotate.

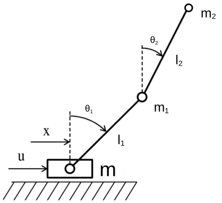


Figure: The Double Pendulum System: from https://cmc.deusto.eus/swinging-up-the-double-pendulum-on-a-cart/

## Outline

## Motivation

The double pendulum is a popular tool in academic settings because it illustrates the intricate, chaotic, output of a seemingly simple physical system.

It has been studied as a machine learning problem and with more traditional robotics control methods such as feed forward control. Machine learning methods are of particular importance because more complex chaotic systems do not always have such straightforward control schemes and therefore more flexible methods are desired.

## Outline

## Single Pendulum vs. Double

Compared to a single pendulum system, which follows a predictable easy to follow path, the equations which control the motion of the double pendulum system are significantly more complex, and unpredictable.

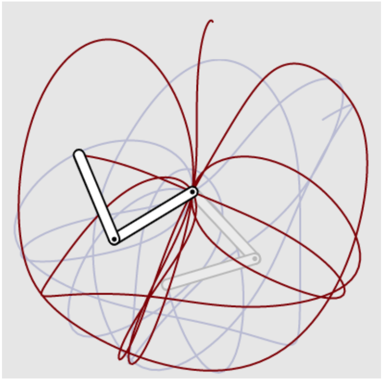


Figure: https://www.complexity-explorables.org/explorables/double-trouble/

## Hold Up Position

Our goal is to create and control a DPOAC system and achieve the following:

- Start the system near a 'Hold up position' (an unstable equilibrium position, specifically where both pendulums are upright), move the pendulums, via applying force to the cart, to that position and keep them there.

## Deriving the Equations of Motion using the Lagrangian

The notation here represents the following $\dot{\theta} = \frac{d\theta}{dt}$ where t is time, and $\ddot{\theta} = \frac{d^2\theta}{dt^2}$, and so on, where $\dot{\theta}$ and $\theta$ and all such parameters are treated as independent of each other when the derivative of the Lagrangian with respect to a such a parameter is taken. The Lagrangian of a system is a function defined as the kinetic energy (T) minus the potential energy (V),

$$L = T - V$$

Taking the following derivatives will give us the equation of motion with respect to variable $\theta$ :

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\theta}}\right) - \frac{\partial L}{\partial \theta} = q$$

Where q is the vector of generalized forces. This is the Euler-Lagrange equation.

## Why the Lagrangian?

The Euler-Lagrange equation is equivalent to Newton's Laws of Motion, but is better suited to more complicated scenarios since it has the same form for any system with generalized coordinates. In this case, we use the angles of the pendulums from the line of rest.

Newtons formalism needs directions for its forces do vectors are required, however the Lagrangian only deals with scalar quantities since it deals with Kinetic and Potential energies.

## Continued

The parameters defining the system are as follows:

Motion Parameters

- $\theta_0 =$ the horizontal position of the cart
- $\theta_1$=the angle created by the rod of pendulum 1
- $\theta_2$=the angle created by the rod of pendulum 2

Constants

- $m_0 =$ mass of the cart
- $l_1$ =pendulum 1 length
- $m_1$ =end mass of pendulum 1
- $l_2$ =pendulum 2 length
- $m_2$ =end mass of pendulum 2

## Continued

The parameters for which we apply the previous function and derive the equation of motion are $\theta_0$, $\theta_1$ , and $\theta_2$.The only force applied to the system is $u(t)$ which is applied horizontally to the cart giving the following system of equations:

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\theta}_0}\right) - \frac{\partial L}{\partial \theta_0} = u(t)$$

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\theta}_1}\right) - \frac{\partial L}{\partial \theta_1} = 0$$

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\theta}_2}\right) - \frac{\partial L}{\partial \theta_2} = 0$$

which is represented as the vector of forces $\vec{q} = [u(t), 0, 0]^T$

## Derivation

For our system the Kinetic energy is

$$T_{tot} = T_{cart} + T_{m_1} + T_{m_2}$$

and the Potential energy is

$$V_{tot} = V_{m_1} + V_{m_2}$$

(potential energy of the cart is zero since it cant move vertically).

The kinetic energy of a moving object is defined as$\frac{1}{2}mv^2$, and its potential energy is $mgh$.

## Derivation

Once the Euler-Lagrange Equations have been obtained for $\theta_0$, $\theta_1$, and $\theta_2$, the result is a non-linear second order system that we can further convert to a first-order problem. It's this first order system which will be translated into Python Code and solved via the following methods outlined during our presentation.

## Outline

## 1st order Runge Kutta

basic idea: know where you are and where you are going ie. the initial conditions, then you can estimate where next you will be.
We can generalize this type of estimation with:

$$y_{n+1} = y_n + \phi \cdot h$$

where $\phi$ is just some estimate of the slope and h is the time step.
Over a time step, the slope is changing, but if we take a small enough time step we can get a fairly good estimate of the function by naively taking $\phi$ to by $\frac{dy}{dx}$ at that point.

## Consider the IVP

$$\frac{dy}{dx} = 2x,\ y_0 = 0,\ h = \frac{1}{2}$$

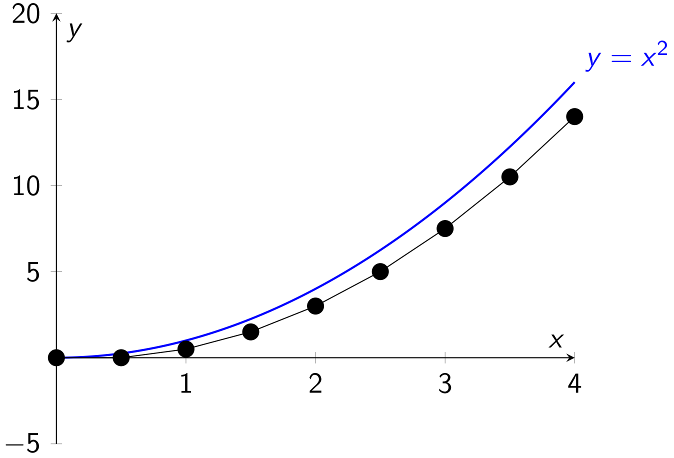
**Step 1**

$$\begin{aligned} x_1 &= \frac{1}{2} \\ y_1 &= y_0 + h \cdot f(x_0) \\ &= 0 + \frac{1}{2} \cdot f(0) \\ &= 0 + \frac{1}{2} \cdot 0 \\ &= 0 \end{aligned}$$

**Step 2**

$$\begin{aligned} x_2 &= 1 \\ y_2 &= y_1 + h \cdot f(x_1) \\ &= 0 + \frac{1}{2} \cdot f(\frac{1}{2}) \\ &= 0 + \frac{1}{2} \cdot 1 \\ &= 0.5 \end{aligned}$$

## example



$$\frac{dy}{dx} = 2x, \qquad y(0) = 0$$

## 4th Order Runge Kutta

take a more sophisticated estimate for $\phi$ as a linear combination of estimates

$$\phi(x, y, h) = a_1 k_1 + a_2 k_2 + a_3 k_3 + ... + a_n k_n$$

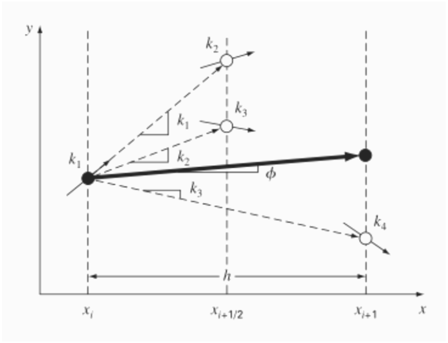


Figure: https://tpaschalis.me/rk4-explained/

$$
\begin{aligned}
k_1 &= hf(t_n, y_n) \\
k_2 &= hf(t_n + \frac{h}{2}, y_n + \frac{k_1}{2}) \\
k_3 &= hf(t_n + \frac{h}{2}, y_n + \frac{k_2}{2}) \\
k_4 &= hf(t_n + h, y_n + k_3) \\
y_{n+1} &= y_n + \frac{k_1 + 2k_2 + 2k_3 + k_4}{6}
\end{aligned}
$$

## why use 4th order Runge Kutta?

- These iterative methods are fairly accurate but require a small step size which gets computationally very expensive. We get a better estimate of the slope with 4th order Runge Kutta so we can take a smaller step size.
- One draw back is that errors compound
- Another drawback is that if the function changes very rapidly the estimation of the function can get off track badly.

## Outline

## Why Reinforcement Learning

- Normal machine learning algorithms require a lot of data to train on, for problems like ours it may be hard to gather effective data
- Reinforcement learning helps eliminate the need for data collection almost entirely by allowing the algorithm to learn from its mistakes
- Q-Learning is one example of the many reinforcement learning algorithms and the one we decide to apply to this problem for a few reasons.

## Overview

Markov decision process (MDP) is a framework to model decision making of dynamic systems. It is comprised of:

- a set of states $S$
- a set of actions $A$, taken in any state
- a set of reward $R$ for the state action pair (s,a)

We call the decision maker an agent and the agent's goal is to maximize the total reward over time.

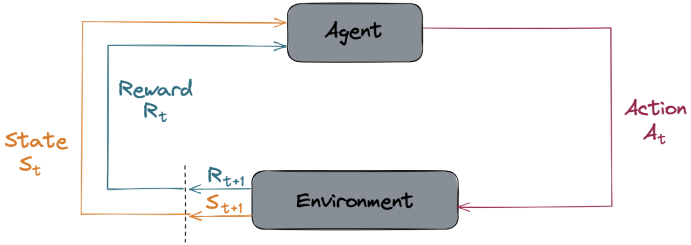
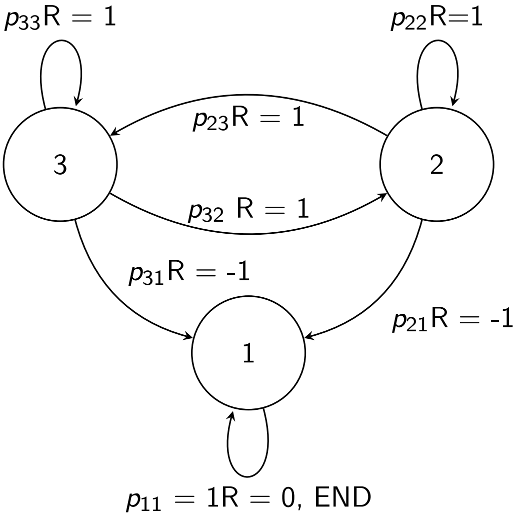


Figure: https://deeplizard.com/learn/video/my207WNoeyA

## MDP example



## Rewards

- We define discount rate $0 < \gamma < 1$, $\gamma \in \mathbb{R}$ such that the expected discounted sum of rewards $G$ at time $t$,

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

  This is so immediate reward has more influence on agent's action. It also ensures the function is finite in an infinite timestep situation.

## Policy and Value Functions

- For each state $s \in S$, a policy, denoted $\pi$, is a probability distribution over actions $a \in A(s)$.
- Action-value function under policy $\pi$, is defined $Q_\pi(s, a) = E[G_t | S_t = s, A_t = a]$ where $Q_\pi$ is the discounted expected return from starting at state $s$ at time $t$ then taking action $a$
- an optimal value function is the max of all possible value functions under any policy $\pi$, denoted $Q_* = \max_\pi Q_\pi$

## Bellman's Optimality Equation

A property of $Q_*$ is that it must satisfy the following equation

$$Q_*(s, a) = E[R_{t+1} + \gamma \max_{a'} Q_*(s', a')]$$

- $R_{t+1}$ is the reward for performing the State action pair $(s, a)$
- $\gamma \max_{a'} Q_*(s', a')$ is the maximum discounted return obtainable from all possible actions in the state you end up in.

## Overview

Q-learning is a reinforcement algorithm with some of the following important properties:

- Attempts to learn an "optimal" policy to a MDP
- Fills and updates a Q-table
- Optimal Q-table leads to optimal policy
- A few Weaknesses

## Exploitation and Exploration

Exploitation and Exploration are both critical concepts in choosing actions for the Q learning algorithm while it is training.

- Exploitation:

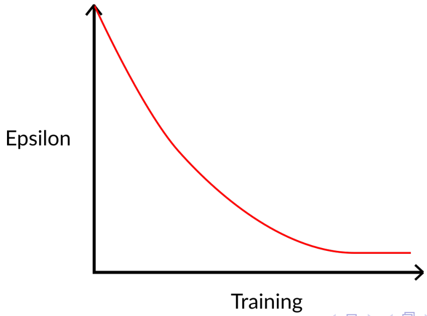
  Using the current best policy to make the greediest action.

- Exploration:

  Randomly take actions with equal probability between the actions available in current state.

## Exploitation VS Exploration

- The $\epsilon$ value definition:
  A real value between 0 and 1 initialized at $\epsilon = 1$
- Using the $\epsilon$ value to determine action:
  Choose an exploration based random action with a probability of $\epsilon$
  Choose an exploitation based greedy action with a probability of $1 - \epsilon$
- Updating $\epsilon$ over time:
  In most implementations $\epsilon$ has an inverse relationship with training such as:



## Q-Table

- The Q or Quality Table is the most important part of Q-learning it is how we derive our policy and what the algorithm is training to refine.
- The Q-Table maps all possible state action pairs to real q values: $Q(S, A) = q_{S,A}$ for a state $S$, action $A$ and $q_{S,A} \in \mathbb{R}$
- The Q table first needs to be filled with placeholder values so that it can then be iteratively updated

## Training the Algorithm

Now that we know what the Q-Table is what does the process of "training" look like for a Q learning algorithm?

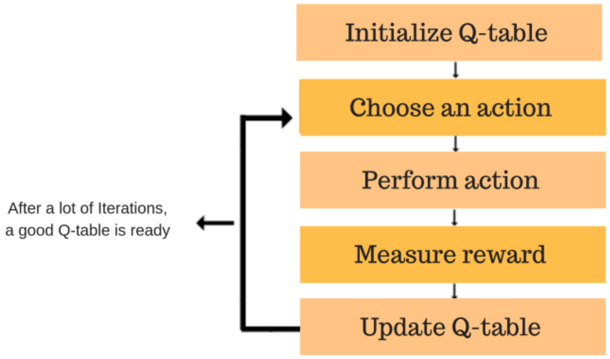


Figure: Method for training

## Updating the Q-Table

To update the Q-Table the following formula modified bellman equation is used:

$$Q(S, A) = (1 - \alpha)(Q(S, A)) + \alpha\{R_{t+1} + \gamma \mathrm{Max} Q'(S', A')\}$$

It will be broken down in the next slide.

## Understanding the Modified Bellman

$$Q(S,A) = (1-\alpha)(Q(S,A)) + \alpha\{R_{t+1} + \gamma \mathrm{Max}_{A'} Q'(S',A')\}$$

- $\alpha$ is the "Learning Rate"
- $Q(S,A)$ is the q value of the state action pair being updated.
- $R_{t+1}$ is the reward for performing the State action pair
- $\gamma$ is the discount factor mentioned earlier
- $\mathrm{Max}_{A'} Q'(S',A')$ is the maximum q value obtainable from all actions in your state you end up in.

## A Simplified Example Intro

The next few slides will guide us through a simple example of a single pendulum on a cart and walking through a step by step guide of how a Q-learning algorithm would compute a few iterations of the Q table.

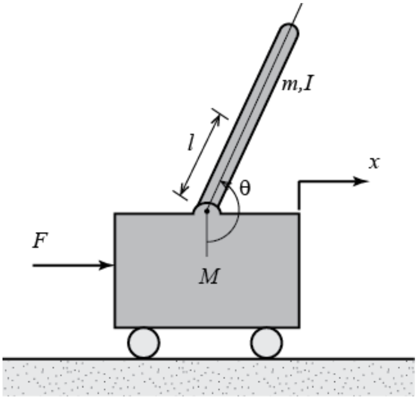
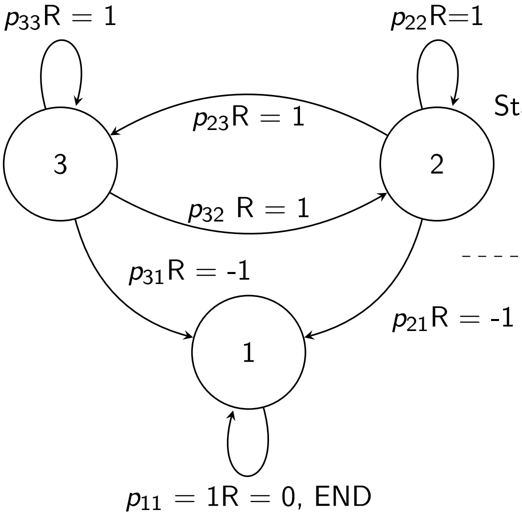
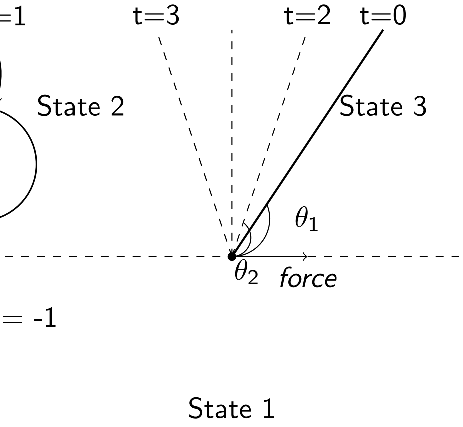


Figure: The system of the example

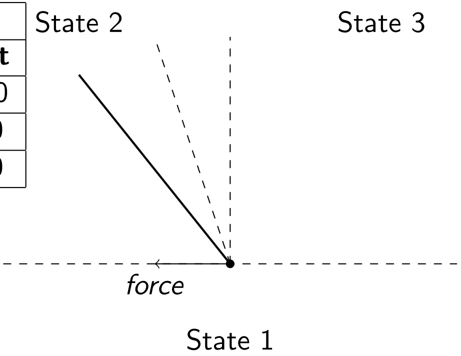
# A Simplified Example





update rule: $Q(s,a) \leftarrow Q(s,a) + \alpha[R + \gamma \max_a Q(s',a) - Q(s,a)]$
$\alpha = \gamma = \frac{1}{2}$

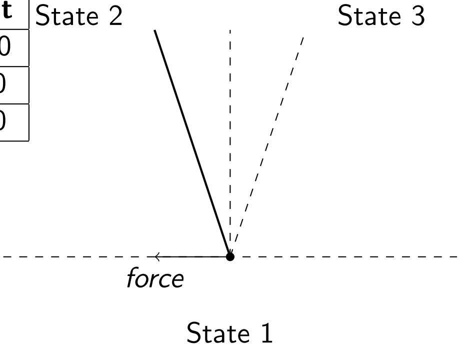
| **State** | **Action** | |
|---|---|---|
| | **Force Left** | **Force Right** |
| **State 1** | $Q(1,L) = 0$ | $Q(1,R) = 0$ |
| **State 2** | $Q(2,L) = 0$ | $Q(2,R) = 0$ |
| **State 3** | $Q(3,L) = 0$ | $Q(3,R) = 0$ |



start in state 2, force to the left, run simulation, end in state 2:
$Q(2,L) = Q(2,L) + \alpha[R + \gamma \max_a[Q(2,L), Q(2,R)] - Q(2,L)]$
$Q(2,L) = 0 + \frac{1}{2}[1 + \frac{1}{2} max(0,0) - 0]$
$= \frac{1}{2}$

update rule: $Q(s,a) \leftarrow Q(s,a) + \alpha[R + \gamma \max_a Q(s',a) - Q(s,a)]$
$\alpha = \gamma = \frac{1}{2}$

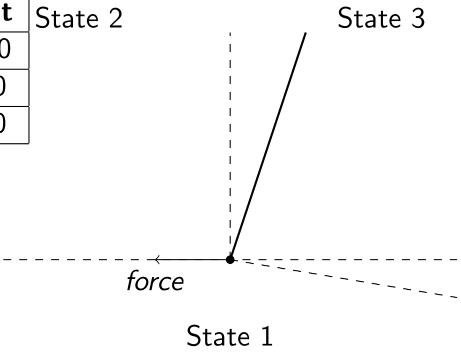
| **State** | **Action** | |
|---|---|---|
| | **Force Left** | **Force Right** |
| **State 1** | $Q(1,L) = 0$ | $Q(1,R) = 0$ |
| **State 2** | $Q(2,L) = \frac{1}{2}$ | $Q(2,R) = 0$ |
| **State 3** | $Q(3,L) = 0$ | $Q(3,R) = 0$ |



start in state 2, force to the left, run simulation, end in state 3:
$Q(2,L) = Q(2,L) + \alpha[R + \gamma \max_a[Q(3,L), Q(3,R)] - Q(2,L)]$
$Q(2,L) = \frac{1}{2} + \frac{1}{2}[1 + \frac{1}{2} max(0,0) - \frac{1}{2}]$
$= \frac{3}{4}$

update rule: $Q(s,a) \leftarrow Q(s,a) + \alpha[R + \gamma \max_a Q(s',a) - Q(s,a)]$
$\alpha = \gamma = \frac{1}{2}$

| **State** | **Action** | |
|---|---|---|
| | **Force Left** | **Force Right** |
| **State 1** | $Q(1,L) = 0$ | $Q(1,R) = 0$ |
| **State 2** | $Q(2,L) = \frac{3}{4}$ | $Q(2,R) = 0$ |
| **State 3** | $Q(3,L) = 0$ | $Q(3,R) = 0$ |



start in state 3, force to the left, run simulation, end in state 1, terminate:
$Q(3,L) = Q(3,L) + \alpha[R + \gamma \max_a[Q(1,L), Q(1,R)] - Q(3,L)]$
$Q(3,L) = 0 + \frac{1}{2}[-1 + \frac{1}{2}max(0,0) - 0]$
$= -\frac{1}{2}$

## Outline

## conclusion

- Q-Learning works fairly well for the stabilization problem. Gustafsson[7] sites convergence 70% of the time. In validation the learned policy is able to stabilize the pendulum from starting positions close to zero, however a marked decrease in performance can be seen when the starting angle for both pendulums exceeds 0.4 degrees.
- Q-learning can be used for relatively simply control mechanism. More sophisticated algorithms are needed to solve the swing up problem.

## References

1 Crowe-Wright, Ian J P. "Control Theory: The Double Pendulum Inverted on a Cart." (2018). https://digitalrepository.unm.edu/math_etds/132

2 Taylor, John R. (John Robert), 1939-. (2005). Classical mechanics. Sausalito, Calif. :University Science Books,

3 Sutton, R. S., & Barto, A. G. (2018). Reinforcement learning: An introduction (2nd ed.)

4 "Control Tutorials for MATLAB and Simulink - Inverted Pendulum: System Modeling." Control Tutorials for MATLAB and Simulink - Inverted Pendulum: System Modeling, ctms.engin.umich.edu/CTMS/index.php?example=InvertedPendulum&se

5 "An Introduction to Q-Learning Part 2/2." An Introduction to Q-Learning Part 2/2, huggingface.co/blog/deep-rl-q-part2.

6 "An Introduction to Q-Learning: Reinforcement Learning." freeCodeCamp.org, 3 Sept. 2018, www.freecodecamp.org/news/an-introduction-to-q-learning-reinforcement-learning-14ac0b4493cc.

## References

7 Gustafsson, Fredrik K.. “Control of Inverted Double Pendulum using Reinforcement Learning.” (2016).