Optimal control in RL

July 18, 2023

1 Imports & Constants

```
import numpy as np
import math
from scipy.integrate import odeint
import matplotlib.pyplot as plt
from tqdm import tqdm
```

Constants: $T_{\max} = u_{\max} = 0.5$, $c = 0.1$, $\sigma_0 = 0.01$, $\tau = 1$, $m = l = 1$, $g = 9.8$, $\mu = 0.01$, timestamp $= 500$, $\Delta t = 20/500$

2 Pre-defined Functions

(1) $g(x)$:
- input: a number x
- output: $\frac{2}{\pi}\tan^{-1}(\frac{\pi}{2}x)$

(2) $g^{-1}(x)$:
- input: a number x
- output: $\frac{2}{\pi} \cdot \tan(\frac{\pi}{2}x)$

(3) $G(u) = c\int_0^{u/u_{\max}} g^{-1}(s)ds$
- input: u as control
- output: $c \cdot -4\,\frac{\log(\cos(\frac{\pi}{2}\frac{u}{u_{\max}}))}{\pi^2}$

(4) reinforcement $r(x, u)$:
- input: $x = (\theta, \dot{\theta})$, u as control
- output: $r(x) - G(u) = \cos(\theta) - G(u)$

(5) a fixed set of 12×12 Gaussian functions:
- input: $\mathbf{x} = (\theta, \dot{\theta})$
- prediction is given by $P(\mathbf{x}(t)) = \sum_{i=1}^{144} w_k b_k(\mathbf{x})$ where $b_k(\mathbf{x}) = \exp\left(-\frac{||\mathbf{x}-\mathbf{c}_k||^2}{2\sigma^2}\right)$ where $\mathbf{c}_k$ is the center of k-th gaussian basis function
- create 12 evenly spaced centers c_1 for θ ranging from $-\pi$ to π and 12 evenly spaced centers c_2 for $\dot{\theta}$ ranging from -10 to 10

- initialize basis_functions to a $12\times12\times1 = 144\times1$ array of 0's, $k = 0$
- for $i = 1, ..., 12$:
  for $j = 1, ..., 12$:
    basis_functions$[k] = \exp\left(-\frac{(\theta - c_1[i])^2 + (\dot\theta - c_2[j])^2}{2\sigma^2}\right)$ where $\sigma = 0.1$
    increase the index k by 1
- output: a $12\times12\times1 = 144\times1$ array of basis_functions

(6) pendulum simulation with limited torque: $ml\ddot\theta = -\mu\dot\theta + mgl\sin\theta + T$
First, to simulate this dynamics equation with function `pendulum_dynamics`
- input: the current state $(\theta, \dot\theta)$, the current time t, the current control T
- output: the derivatives $(\dot\theta, \ddot\theta)$ of the current state $(\theta, \dot\theta)$

Next, to solve for this dynamics equation, use the `odeint` library from `python scipy`
- input: the simulation function `pendulum_dynamics`, the current state $(\theta, \dot\theta)$, the current control T, the time period $[0, \Delta t]$
- output: the next state $(\theta, \dot\theta)$ based on the previous state and control

(7) error $\hat r(t)$:
- input: current state $x(t)$, previous state $x(t-\Delta t)$, current weight $v(t)$, current control $T(t)$, $\tau = 0.4$ since assumes that the discount factor $\gamma = 1 - \frac{\Delta t}{\tau} = 0.9$
- output: $\hat r(t) = r(x(t), T(t)) + \frac{\tau}{\Delta t}\left[(1 - \frac{\Delta t}{\tau})P(x(t)) - P(x(t-\Delta t))\right]$

where $P(x(t)) = \sum_{i=1}^{144} v_i b_i(x(t))$ and $P(x(t-\Delta t)) = \sum_{i=1}^{144} v_i b_i(x(t-\Delta t))$

(8) sum of absolute error with the weight v:
- input: a 500×2 array of $x(t)$, weight v (constant), a 500×1 array of $T(t)$
- for $k = 1, ..., 500$, $t = k\Delta t$
  solve for the error $\hat r(t)$ based on the current state $x(t)$, the previous state $x(t-\Delta t)$, weight v, the current control T at time $k\Delta t$
- output: $\sum_{i=1}^{500} |\hat r(i)|$ = sum of abs errors at each of the 500 states

3 Generate Data

Step 1: generate $x(t) = (\theta(t), \dot\theta(t))$ for $t = k\Delta t$ with $\Delta t = \frac{20}{500}$, $k = 1, ..., 500$
- input: $x(0) = (0, 0)$, $T(t) \sim$ Uniform[min $= 0$, max $= 5$] for $t = k\Delta t$
- for $k = 1, ..., 500$, $t = k\Delta t$
  solve for the next state $x(k+1)$ based on the current state $x(k)$, the current control T at time $k\Delta t$, and the pre-defined functions (6)
- output: a 500×2 array of $x(t) = (\theta(t), \dot\theta(t))$ for $t = k\Delta t$

4 Optimal Control

Step 2: Calculate $P(t) = \sum_{i=1}^{144} v_i b_i(x(t))$ where $b_i()$ are the basis Gaussian functions dimension
- input: a 500×2 $x(t)$ array for $t = k\Delta t$, weight v, control T

- initialization: weight v = a 144×1 array of 1's, t_up = 0
- for $i = 1, ..., 500$
    - current prediction $P(t)$ = dot product of current weight v and $b(x(t))$
    - calculate error $\hat{r}(t)$ according to the pre-defined function (7) (Note: if the current state is the initialization state and there is no previous state, the error $\hat{r}(t)$ is set to 0)
    - calculate sum of absolute error w.r.t the current weight v at the i^{th} timestamp using the pre-defined function (8)
    - according to $\Delta v_i \propto \hat{r}(t) b_i(x(t - \Delta t))$, update weight s.t. $v = v + \alpha \cdot \hat{r}(t) \cdot \text{gaussian(prev state)}$ where $\alpha = 0.01$ is the learning rate [new way of updating v]
    - if $\theta(t) < 90°$ or $> 270°$, increase t_up by 1.

- output: a 500×1 array containing the prediction $P(x(t))$ of each $x(t)$, t_up, the final weight v after 500 times update, a 500×1 array of the sum of absolute errors w.r.t each v in each iteration

Step 3: Calculate the derivative $\frac{\partial P(x)}{\partial x}$ with input $x(t)$

Since $P(t) = \sum_i v_i b_i(x(t)) = v_1 b_1(x(t)) + ... + v_{144} b_{144}(x(t))$ and $b_i(x(t))$ is the Gaussian basis, $x_0 = \theta$, $x_1 = \dot{\theta}$

$b_1(x(t)) = \exp\left(-\frac{(x_0-\mu_{0,1})^2+(x_1-\mu_{1,1})^2}{2\sigma^2}\right), ..., b_{12}(x(t)) = \exp\left(-\frac{(x_0-\mu_{0,1})^2+(x_1-\mu_{1,12})^2}{2\sigma^2}\right)$

$b_{13}(x(t)) = \exp\left(-\frac{(x_0-\mu_{0,2})^2+(x_1-\mu_{1,1})^2}{2\sigma^2}\right), ..., b_{24}(x(t)) = \exp\left(-\frac{(x_0-\mu_{0,2})^2+(x_1-\mu_{1,12})^2}{2\sigma^2}\right)$

...

$b_{133}(x(t)) = \exp\left(-\frac{(x_0-\mu_{0,12})^2+(x_1-\mu_{1,1})^2}{2\sigma^2}\right), ..., b_{144}(x(t)) = \exp\left(-\frac{(x_0-\mu_{0,12})^2+(x_1-\mu_{1,12})^2}{2\sigma^2}\right)$

then $\frac{\partial P(x)}{\partial x_0} = v_1 b_1(x(t)) \frac{\partial b_1}{\partial x_0} + ... + v_{144} b_{144}(x(t)) \frac{\partial b_{144}}{\partial x_0}$
$= v_1 b_1(x(t)) \cdot \left(-\frac{x_0-\mu_{0,1}}{\sigma^2}\right) + ... + v_{144} b_{144}(x(t)) \cdot \left(-\frac{x_0-\mu_{0,12}}{\sigma^2}\right)$

Similarly, $\frac{\partial P(x)}{\partial x_1} = v_1 b_1(x(t)) \frac{\partial b_1}{\partial x_1} + ... + v_{144} b_{144}(x(t)) \frac{\partial b_{144}}{\partial x_1}$
$= v_1 b_1(x(t)) \cdot \left(-\frac{x_1-\mu_{1,1}}{\sigma^2}\right) + ... + v_{144} b_{144}(x(t)) \cdot \left(-\frac{x_1-\mu_{1,12}}{\sigma^2}\right)$

Thus, $\frac{\partial P(x)}{\partial x} = \left(\frac{\partial P(x)}{\partial x_0}, \frac{\partial P(x)}{\partial x_1}\right)$

- input: a 500×2 array of $x(t)$, weight v, 12 evenly spaced centers for $x_0 = \theta$ and $x_1 = \dot{\theta}$
- output: a 2×1 array of $\frac{\partial P(x)}{\partial x}$

Step 4: Calculate the optimal control

$\mathbf{b} = (0, 1)^T$, $T^{\max} = 5$, $c = 0.1$, $\tau = 1$, $g(x) = \frac{2}{\pi}\tan^{-1}(\frac{\pi}{2}x)$, $\sigma_0 = 0.01$
- input: final weight v returned in step 2, state x
- for each timestamp t, the optimal control $T(t)$ is given by

$$T = T^{\max} g\left(\frac{T^{\max}}{c}\tau\frac{\partial P(x)}{\partial x}\mathbf{b} + \sigma n(t)\right)$$

where $\sigma = \sigma_0 \cdot e^{P(t)} = \sigma_0 \cdot e^{\sum v \cdot b(x(t))}$ and $n(t) = (N(0,1))$ a random noise. Use the final weight v for the calculation of $\frac{\partial P(x)}{\partial x}$
- output: optimal control $T(t)$

Overall Algorithm with Optimal Control:

```
# initialization
T(t) ∼ Uniform[min=0, max=5]
w = a 144×1 array of 1's

for trial = 1,...,100:
    generate data with the input control T(t) using step 1
    record t_up & the final weight w of this trial using step 2
    record sum of abs error w.r.t w using pre-defined function (8)
    update the input control T(t) for each timestamp t using step 4
    update w with the final weight w
```

5 Actor Critic

Step 2*: Update weights w_i, predictions $= \sum_i w_i b_i(x(t))$ used in actor-critic
- input: a 500×2 $x(t)$ for $t = k\Delta t$, weight w
- for $i = 1, ..., 500$
    - current prediction $P(t) =$ dot product of current weight v and $b(x(t))$
    - calculate error $\hat{r}(t)$ according to the pre-defined function (7) (Note: if the current state is the initialization state and there is no previous state, the error $\hat{r}(t)$ is set to 0)
    - calculate sum of absolute error w.r.t the current weight v at the i^{th} timestamp using the pre-defined function (8)
    - according to $\Delta w_i \propto \hat{r}(t)n(t)b_i(x(t))$, update weight s.t. $w = w + \alpha \cdot \hat{r}(t) \cdot n(t) \cdot \text{gaussian(current state)}$ where $\alpha = 0.01$ is the learning rate and $n(t) = (N(0,1))^{144}$ [144×1 $n(t)$?] a 12×1 random noise [new way of updating v]
    - if $\theta(t) < 90°$ or $> 270°$, increase t_up by 1

- output: a 500×1 array containing the prediction $P(x(t))$ of each $x(t)$, t_up, the final weight w after 500 times update, a 500×1 array of the sum of absolute errors w.r.t each w in each iteration

Step 3*: Calculate the optimal control
- input: final weight w returned in step 2*, state x
- for each timestamp t, the optimal control $u(t)$ is given by

$$u(t) = u^{\max} g\left(\sum_i w_i b_i(x(t)) + \sigma n(t)\right)$$

where $\sigma = \sigma_0 \cdot e^{P(t)} = \sigma_0 \cdot e^{\sum w \cdot b(x(t))}$ and $n(t) = (N(0,1))$ a random noise
- output: a 500×1 array of $u(t)$ (the optimal control for each state)

Overall Algorithm with Actor Critic:

```
# initialization
T(t) ~ Uniform[min=0, max=5]
w = a 144×1 array of 1's

for trial = 1,...,100:
    generate data with the input control T(t) using step 1
    record t_up & the final weight w of this trial using step 2*
    record sum of abs error w.r.t w using pre-defined function (8)
    update the input control T(t) for each timestamp t using step 3*
    update w with the final weight w
```