CS 747: Programming Assignment 3

Ankish Chandresh(24M2163)

April 13, 2025

Optimal Driving Control

Experiments with CMA-ES

I started with experimenting with CMA-ES, successful elements of this approach would be a suitable feature vector extracted from the 13×13 obs, a good fitness function, reasonable function approximation, and hyperparameters of CMA-ES like num_gen, pop_size and num_policy_params.

I chose to use linear function approximation and a feature vector as mean of indices of the first occurrence of a 1 from left and from right for all rows 1 to 13 of the 13×13 obs, if there are no 1 in a row then choose the feature vector value as 6. This way I get a feature vector of size 13×1 storing the mean of indices, I combine this linearly by a learnable weight matrix of size 2×13 and add a bias of size 2×1, note the output of the weight matrix and bias is 2 for two control signals of acceleration and steering and is obtained as np.tanh(W @ feature.flatten()+ b). Also note that my look ahead from the current state is of 13 later in the report a look ahead of 1 suffices.
Example of feature vector for the below obs is [(0+6)/2, (0+6)/2, (1+7)/2, (2+7)/2, (2+7)/2, (2+8)/2, (3+8)/2, (3+8)/2, (3+8)/2, (3+8)/2, (3+8)/2, (3+8)/2, (3+8)/2] = [3, 3, 4, 4.5, 4.5, 5, 5.5, 5.5, 5.5, 5.5, 5.5, 5.5, 5.5]:

Figure 1: Obs

I chose fitness function to be the average distance travelled in 10 episodes, averaged over episodes.

Running CMA-ES with parameters num_gen = 50, pop_size = 10, num_policy_params = $2 \times 13 + 2$, suprisingly gave a policy, under the feature vector being mean of the occurrences of 1 for each row and being linearly combined into control signals, such that the car runs on the middle of the entire road.
This approach however did not clear all test cases.

Experiments with PID Control

Motivated by the observation that CMA-ES solution tried to make the car run in the centre of the entire road, I decided to use some algorithm that minimises this error between the centre of the entire road in the obs and the centre of the car i.e. 6. A natural choice to exercise control for minimising error via some control signal is (Proportional–integral–derivative controller)PID control.
The control function in PID is given by:

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)\, d\tau + K_d \frac{de(t)}{dt}, \tag{1}$$

where $e(t)$ is difference between goal state and current state.

Note that I am breaking down the problem of optimal steering to first trying to optimally drive in the centre of the entire road and then to accelerate suitably to cover the track quickly while not running in to the boundary of the road.

I have used a special case of PID namely PD control where the K_i parameter is set to zero. In order to implement PD control I have implemented a PD class with parameters K_p, K_d, goal, last_error. The *observe*() function of this class gives the required control signals. The output signal $u(t)$ is squashed by tanh non-linearity to keep the control signal between -1

and 1. The goal is set to 6 representing the centre of the two lanes(i.e. mean of indices of the occurrence of first 1 from left and then from right for just the last row of obs) must lie at 6 in obs. Note in setting goal to 6 I am only looking ahead by one step into the future out of the 13 step look ahead. The observe function has one more detail i.e. how to find the current_state used to calculate the error(goal - current_state), the current_state or the current centre of the two lanes is the set to the mean of position of the first occurrence of a 1 from left and first occurrence of a 1 from the right in the 13th(last) row. In the code below feature1 is the first occurrence of a 1 from left, feature2 is the first occurrence of a 1 from right looking ahead by one step and x(current_state) is the mean of these two positions rounded to integer. Example computation of x with respect to **figure 1** is mean of occurrences of first 1 from left and right for the last row i.e. x = (3 + 8)/2 = (int)5.5 = 5. If there is no 1 in the last row x = 6.

```
feature1 = 0
feature2 = 12
for i in range(12, 13):
    ok = 0
    for j in range(13):
        if(obs[i][j] == 1):
            ok = 1
            feature1 = j
            break
    if(not ok):
        feature1 = 6

for i in range(12, 13):
    ok = 0
    for j in range(13):
        if(obs[i][12 - j] == 1):
            ok = 1
            feature2 = ((12 - j))
            break
    if(not ok):
        feature2 = 6

x = (int)(feature1 + feature2)/2
```

I noted for a large range of values of K_p, and K_d I could achieve car driving in the centre of the road. For smaller values of K_p, and K_d the car runs more smoothly compared to large K_p, and K_d where it gets influenced much more sharply. I chose it to be set K_p, and K_d at 0.7 and 0.7 respectively to balance smooth and no steering as for small K_p, and K_d there might be no steering at all.

This PD control finally achieves driving in the centre of the entire road, for acceleration I tried hard coding the contol as $(1 - 2 \times np.abs(steering))$. This is done to ensure that when

abs(steering) is large the car should deaccelerate and accelerate otherwise. This approach however doesn't work when starting states have a non-zero large steering causing the vehicle to not move at all due to negative acceleration in the beginning.

For no acceleration the car runs at a speed of 5 which makes the car not hit the edge of the tracks for the entire episode but covers a small distance due to low speed and no acceleration. Hence, we have effectively made some progress to clear all test cases, now what remains to do is to do acceleration control.

Reference for PD: `statusfailed.com/blog/2022-12-08-pid-controller-in-the-openai-gym`

Experiments with PID Control and CMA-ES

Here I tried a hybrid approach where driving in the centre of the two lanes is controlled by PID or PD control and the acceleration control is done by CMA-ES by choosing weight to of size 1×13 and bias 1×1 and feature vector as described in above sections for CMA-ES. I did many runs of CMA-ES under different seeds, I got some success with this hybrid approach but the final scores didn't clear all test cases.

Final Solution

After trying out all the above experiments, I finally concluded that PD controller for driving in the centre of the lane and achieving a target speed of 10 via another PD controller for acceleration works best for all public test cases.
The modified PD class has the following parameters: K_p, K_d, K_p_v, K_d_v, goal, v_goal, last_error, last_v_error. Where subscript v denotes control parameters for velocity and other parameters are as described above. The below code implements returning of the control signals:

```
x = (int)(feature1 + feature2)/2
error = self.goal - x

d_error = error - self.last_error
self.last_error = error
X = -1*np.tanh(self.kp * error + self.kd * d_error)

v_error = self.v_goal - info["speed"]
d_v_error = v_error - self.last_v_error
self.last_v_error = v_error
Y = (1-np.abs(X))*np.tanh(self.kp_v * v_error + self.kd_v * d_v_error)
return [Y, X]
```

Note that x is as in previous sections, goal is 6 and v_goal is 10 and info["speed"] gives the current speed. Note that in order to couple the steering control X and modified acceleration control Y, acceleration signal is multiplied by (1 - np.abs(X)) or (1 - np.abs(steering)) for

driving at speed 10 so that when steering signal X is large the modified acceleration signal Y is small and vice versa. Also note that X and Y are outputs of control signal squashed by tanh. error variable stores the current goal - cuurent_state and d_error stores the derivative of the error similarly for velocity.

With kp=0.7, kd=0.7, kp_v=0.5, kd_v=0.5, goal=6, v_goal = 10. I get all public test cases passed.