Learning State Machines from Neural Policies for F-16 GCAS

Abstract

This work tries to develop and test a method for synthesizing state machines from neural policies for the F-16 GCAS task. On a high level it modifies PROPEL using Jeevana's method, to synthesize state machines, instead of a programmatic policy with a pre-specified DSL. Code Link

Algorithm

1. Learn neural policy f
2. h := f
3. Distill h into state machine policy g
    a. DAGGER:
        i. Iteratively generate trajectories from h
        ii. Update g (Using Jeevana's algorithm)
4. Lift g, (linear combination with f) h := g + \lambda f
5. Neural updates:
        i. f: = Train_via_DRL(h) [take actions from h, apply gradient updates on f]
6. h := g + \lambda f
7. Go to Line 3

Neural Policy

The states(observation space) and actions for the 2-layer neural network:

states: [vt, alpha, beta, phi, theta, psi, p, q, r, pn, pe, h, power]
actions: [Nz_ref, ps_ref, Ny_r_ref, throttle_ref]

The following initial conditions were used for training the agent:

```
Vt = [2000, 2500]
alpha = [-5, 5]deg
theta = alpha
phi = [-5, 5]deg
psi = [-5, 5]deg
Theta_dot, phi_dot, psi_dot = [-10,10]deg/s
pn = pe = 0
Altitude = [20000, 30000]
power = 20
```

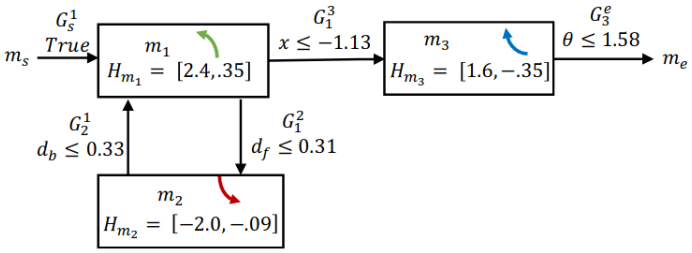
```
Stopping condition: altitude is within [800, 1500] ft and pitch
angle is within [0, 30] deg
```

Many reward functions were tried:

1. -5 on every time step, -10000 on crashing
2. -5-k1*(altitude deviation from 1000) on every time step, -10000 on crashing
3. -5-k2*(pitch deviation from 0) on every time step, -10000 on crashing
4. -5-k1*(altitude deviation from 1000)-k2*(pitch deviation from 0) on every time step, -10000 on crashing

The last reward function performed the best, 72% trajectories pass when initial conditions are randomly sampled from the above values.

State Machine



Example State Machine

Each state(or mode) of the state machine has action parameters p, which define the action in that mode based on the current observations, i.e. action = f(observations, p). For each parameter in p, the algorithm learns two values, mean and std dev. Then to arrive at a state machine, we can sample from the normal distribution with this mean and std dev. The switching conditions are conditions on observations (e.g obs[1] > 0.5), and can be conjoined by logical AND (e.g obs[1] > 0.4 & obs[3] <0.1). The maximum number of such terms conjoined by logical AND has to be pre-specified in the algorithm(decision tree depth). The algorithm learns mean and std dev for these threshold values in the inequalities.

Results using PROPEL+Jeevana's approach (as in the above algorithm)

states: [vt, alpha, beta, phi, theta, psi, p, q, r, pn, pe, h, power]
actions: [Nz_ref, ps_ref, Ny_r_ref, throttle_ref]

The following state machine is derived after 20 iterations.

Ny_r_ref is always set to 0, so we predict only 3 actions

[[[-1.110161666335307], [-1.6615664780084436], [0.6615664780084436]],
[[3.093678234872306], [0.596986153180812], [0.431562345800854377]],
[[4.5252976114874051], [1.461928973849725], [0.36615664780084436]],
[[2.2718556639249257], [2.3272199958537267], [0.5615624480085556]]

[[0.37093611367055546, 0.32245784842859126, 0.32245784842859126],
[0.6479855782951838, 0.5039902961886502, 0.32245784842859126],
[2.214705283061544, 2.2344420048739213, 0.32245784842859126],
[0.33013133536453776, 0.3739238609898631, 0.32245784842859126]]

{-1: {0: 'LinearCond([0.0, -1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, -0.12667825645048017])',
1: 'LinearCond([0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, -1.0, 0.07085774437009962])',
3: 'LinearCond([0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, -0.11554231344486408])',
 2: 'LinearCond([0.0, 0.0, 0.0, 0.0, -1.0, 0.0, 0.0, 0.0, -0.11898796691462205])'},
1: {0: 'AndCond(LinearCond([0.0, 0.0, -1.0, 0.0, 0.0, 0.0, 0.0, 0.0,
-0.7297520221675076]),LinearCond([-1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.9239791339602537]))',
3: 'AndCond(LinearCond([0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0,
-2.4209807769388547]),LinearCond([-1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
-448.805330153875]))',
2: 'AndCond(LinearCond([0.0, 0.0, 0.0, 0.0, -1.0, 0.0, 0.0, 0.0,
-1.1524095089599566]),LinearCond([0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0,
-1.5834492666215674]))'},
2: {0: 'AndCond(LinearCond([1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
-0.2586460118071705]),LinearCond([0.0, 0.0, 0.0, 0.0, 0.0, -1.0, 0.0, 0.0,
0.2697645259001959]))',
3: 'LinearCond([1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1062.6227983997348])', 1:
'AndCond(LinearCond([0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0,
-0.6432479386427706]),LinearCond([0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, -1.0,
2.1208083560253757]))'},
0: {1: 'LinearCond([0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, -0.7903313058467776])', 2:
'LinearCond([0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, -1.4856965973215388])', 3:
'AndCond(LinearCond([0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0,
-0.4816898057675454]),LinearCond([0.0, 0.0, 0.0, -1.0, 0.0, 0.0, 0.0, 0.0,
-0.01960514613204367]))'},
3: {1: 'AndCond(LinearCond([0.0, 0.0, 0.0, 0.0, -1.0, 0.0, 0.0, 0.0,
-0.5050759776362947]),LinearCond([0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0,
0.5901763465427293]))',
0: 'AndCond(LinearCond([0.0, -1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.3929616670224201]),LinearCond([0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
-0.5151818474709313]))',
2: 'AndCond(LinearCond([0.0, 0.0, 0.0, -1.0, 0.0, 0.0, 0.0, 0.0,
-0.5681830636809332]),LinearCond([0.0, 0.0, 0.0, 0.0, -1.0, 0.0, 0.0, 0.0,
-1.2528497011995905]))'}}
{-1: {0: '0.18707629673710507',
1: '0.3465970144427055',

3: '0.019177231874170825', 2: '0.19868693484739486'}, 1: {0: '153.98791078074566', 3: '54.231144303029716', 2: '23.474669327757546'}, 2: {0: '155.00205237659947', 3: '4.2024932089591704', 1: '20.72205884909231'}, 0: {1: '226.28355957149327', 2: '123.28165986433127', 3: '122.27513108342099'}, 3: {1: '118.09558251483203', 0: '45.52683776497025', 2: '11.643122315713505'}}

The way to interpret this state machine: Each mode has action parameters which are described by the mean and std dev of the random variable corresponding to it. The first line is the mean, the second line is the std_dev. So for the above state machine,

Mode_1:

Nz_mean = -1.110161666335307 , Nz_std = 0.37093611367055546
ps_mean =-1.6615664780084436 , ps_std = 0.32245784842859126
throttle_mean = 0.6615664780084436, throttle_std = 0.32245784842859126

Similarly for other modes. The switching conditions are to be interpreted as follows.

0: {1: 'LinearCond([0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, -0.7903313058467776])'

To transition from state 0 to 1, obs[3] > -0.7903313058467776

0: {1: 'LinearCond([0.0, 0.0, -1.0, 0.0, 0.0, 0.0, 0.0, 0.0, -0.7903313058467776])'

To transition from state 0 to 1, obs[3] < 0.7903313058467776.

We also have AND conditions, which are basically linear conditions conjoined by logical and. The parameters for switching conditions are also normal random variables, the conditions themselves represent the mean, and the std_dev of the parameters follows the conditions.