Project #1 - JAX Controller

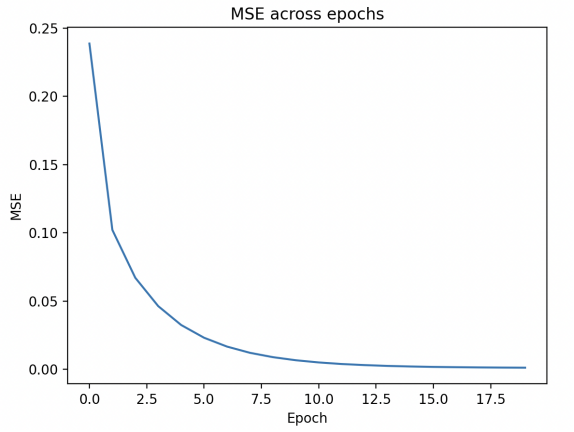
Table of contents

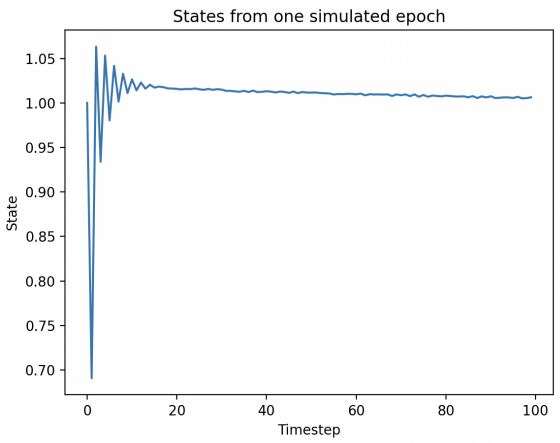
Bathub plant

NN controller

This run shows a neural net quickly being able to correct the errors in the bathtub plant. This simulation is also ran with avgE instead of sumE, leading to quite a high learning rate and intial values in the NN. ReLU activation function is also used, this implies that U cant be negative. But this works well since the controller is trying to ADD water to keep the height stable. The reason i changed the integral part to an average part was because the sumE feature got way larger than the others, especially if the starting error is quite big.

Plant	Bathub	A = 10, C=0.7, H0 = 1
Controller	NN	ReLU, layers=[4,4], range=[0.0,1.5]
Epochs	30	
Timesteps	100	
Lr	0.3	

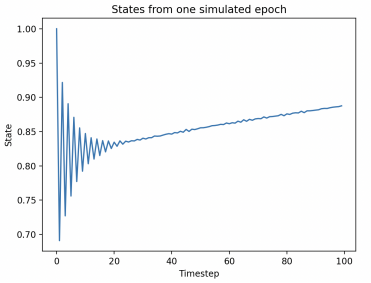


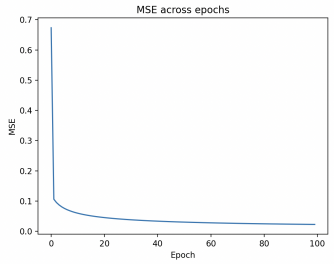


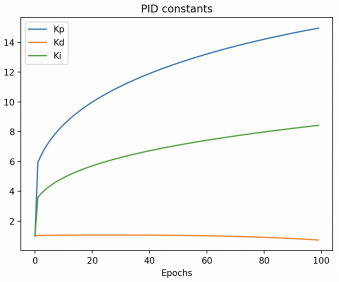
PID controller

This run shows how the constants evolve when set to values way lower than what they should be. The integral part of the controller plays a big part in the bathtub problem, but when changed from sumE to avgE, which wil make the size more similar to the other params, we can see that Kp plays a big part in the result.

Plant	Bathub	A = 10, C=0.7, H0 = 1
Controller	PID	Kp = 1.0, Kd = 1.0, Ki = 1.0
Epochs	100	
Timesteps	100	
Lr	20	“Extremely high because sumE change to avgE, this lead to really small gradients compared to the desired parameter values.”





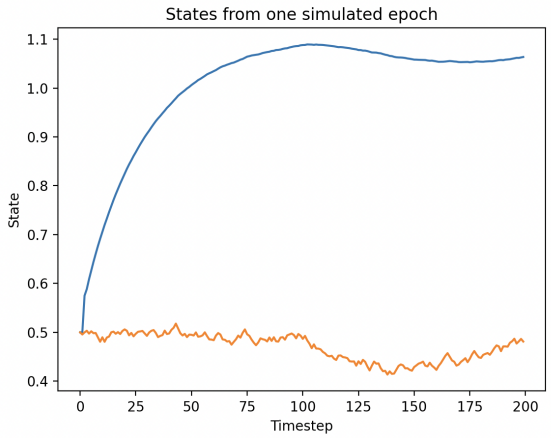


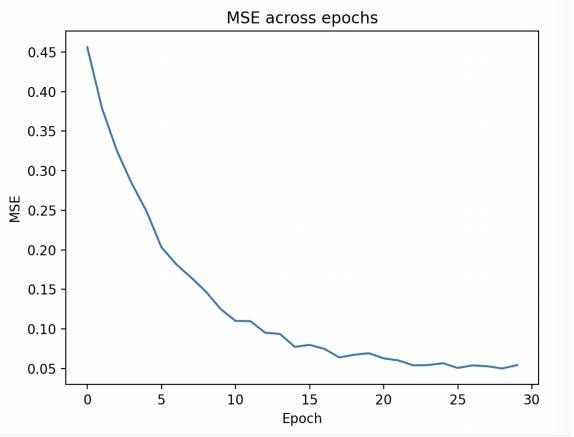
Cournot competion

NN controller

Here we have a slow learning NN finding a stable profit for the cournot competition problem. This is quite a slow progression, so one could probably find a solution increasing the production c_1 faster. Ex by using a learning rate scheduler or finding a better range of initial parameters.

Plant	Cournot	p_max=4, c_m=0.1, target=2.5, c_1 = 0.5, c_2 = 0.5
Controller	NN	Tanh, layers=[4,4], range=[-0.35, 0.35]
Epochs	30	
Timesteps	200	
Lr	0.0005	

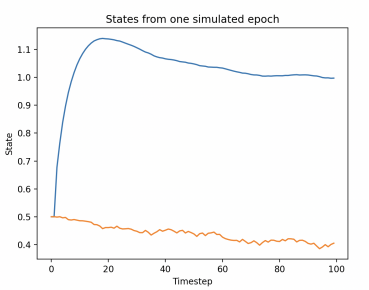


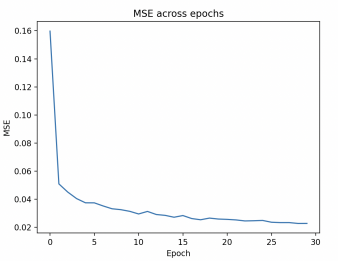


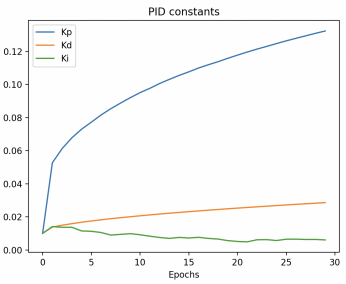
PID controller

Here we can observe the PID controller starting with constants close to zero, and quickly evolving to adjust to the target profit.

Plant	Cournot	p_max=4, c_m=0.1, target=2.5
Controller	PID	Kp = 0.01, Kd = 0.01, Ki = 0.01
Epochs	30	
Timesteps	100	
Lr	0.01	







Tailgating Plant

This plant is supposed to model two cars, where one car starts behind the other, and has to catch up as quily as possible, then tailgate the car at a target distance.

v : velocity of car 1
u : velocity of car 2
d : distance from car 1 to car 2

Updates as follwing:

$$v_{t+1} = U + v_t$$
$$u_{t+1} = D + u_t$$
$$\frac{\partial d}{\partial t} = u_t - v_t$$
$$d_{t+1} = d_t + \frac{\partial d}{\partial t}$$

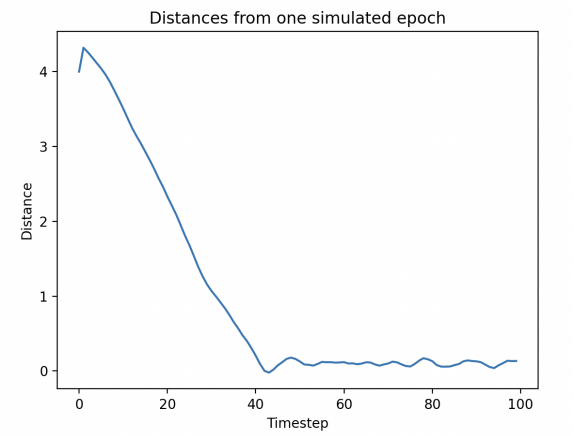
where U is the controller output and D is the disturbance.

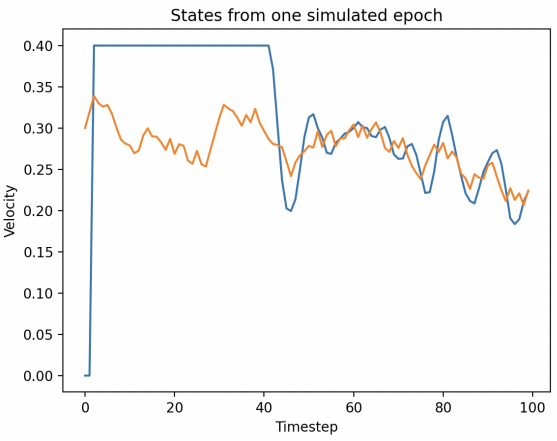
PID controller

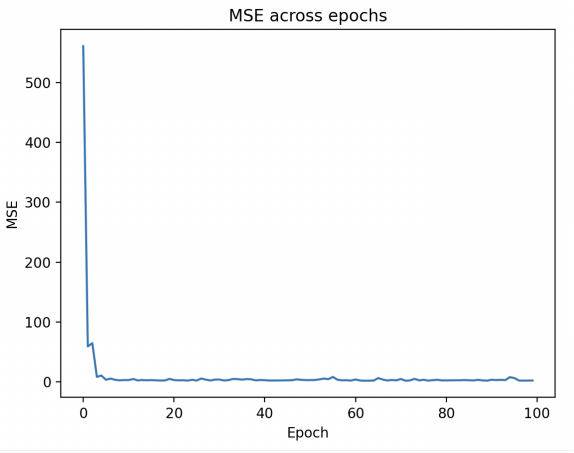
With this plant, the starting MSE will be very high. Therfore i had to turn to an adaptive learning rate which increases for each epoch. I could also have tried a cosine scheduler or a similar scheduler, giving a decay in when the gradient descent algorithm hopefully has discovered a local minimum. We can observe that the car starts following the car ahead, trying to stay right behind the car by adjusting the velocity. The behaviour is unpredictable, so it is natural that the network cant follow exactly behind the car, but we can see that the car adjusts its velocity to compansate for being to far ahead or behind the car.

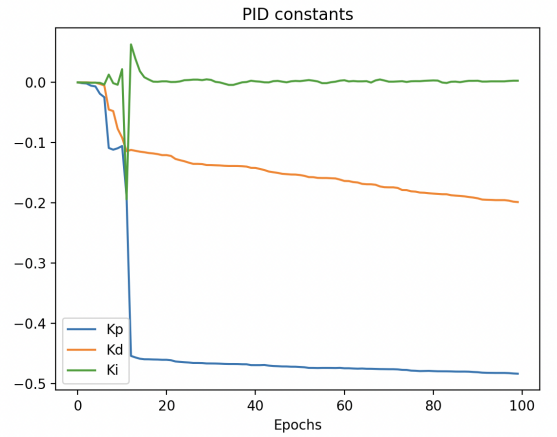
Plant	Tailgating	v_1=0, v_2=0.3, d_0=4, d_target=0.1
Controller	PID	Kp = 0.0001, Kd = 0.0001, Ki = 0.0001

Epochs	100	
Timesteps	100	
lrate_init	1.0 * 10^-9	
lrate_max	0.1	









NN controller

This was really hard to get to work, it seems like if you add any layers, the system starts over-complicating the outputs, and making Asin(x)*e^-(Bx) curves with different amplitudes (A) and decays (B). Changing the NN to no hidden layers gives a better output, but this is basically the same as using the PID controller with one more Bias term. Adding a speed limit also caused huge problems, with the NN controller not being able to adjust for this.

Once the network gets multiple layers deep, the bias term of the output neuron explodes, dominating all other gradients and causing the network to just predict the same output for

every input. I tried to adujst the learing rate during the run to compensate for the high inital error, but it is still not able to find a good solution to the problem.

Plant	Tailgating	v_1=0, v_2=0.3, d_0=4, d_target=0.1
Controller	NN	Tanh, layers=[6], range=[-0.1,0.1],
Epochs	20	
Timesteps	100	
lrate_init	1.0 * 10^-6	
lrate_max	1.0 * 10^-6