Q-Learning for Speed

Kelton Busby
busbykt@mail.uc.edu

Abstract

In this report, Q-Learning is used to maximize velocity and minimize driver fatigue in a human powered vehicle attempting a land speed record. Q-Learning results in a set of Q-values for state-action pairs, called a policy. That policy is utilized here to provide a function of power input over time for a given driver, vehicle, and environment combination that maximizes velocity for the driver's fatigue limit. This function of power input over time results in higher achievable speeds than other methods of developing a power over time function.

1 Problem Background

A yearly competition to break a human powered land speed record takes place in Battle Mountain, NV. Competitors attempt to accelerate their vehicle over a 5 mile stretch of flat road towards a 200 meter section where their time is recorded, and a speed is calculated. The record for that speed as of March 2021, is 89.59 miles per hour.

`http://ihpva.org/hpvarecl.htm#nom01`

To reach high speeds the driver must manage fatigue while accelerating. Maximizing velocity while minimizing fatigue ensures that as much energy as possible is available to the driver to reach a highest possible speed. So how should one go about managing fatigue? The driver's only control is how much power to input to the pedals. Input power is a combination of force on the pedals and cadence or revolutions per minute (RPM). The result of this input is accumulation of fatigue, and acceleration of the vehicle. To maximize velocity, the driver must gain velocity while minimizing fatigue accumulation. As velocity increases, the driver must input more power to accelerate, which increases fatigue. The optimal policy of power input over time is sought using Q-Learning.

2 Governing Physics

Most vehicles attempting this record are lightweight aerodynamic shells that roll on bicycle-like wheels, pedaled by a person in a recumbent position. A vehicle rolling on flat ground requires power to overcome rolling resistance, aerodynamic drag, and accelerate. The power applied to overcome those resistances and accelerate is reduced by mechanical losses within the drivetrain of the vehicle. The power equation is:

$$P_{total} = \frac{P_R + P_D + P_A}{\eta}$$

Where P_R is power to overcome rolling resistance, P_D is power to overcome aerodynamic drag, P_A is power to accelerate, and η is drivetrain efficiency. Each power on the right hand side can be broken down further into its constituent parts:

$$P_R = vmgC_{rr} \qquad P_D = \tfrac{1}{2}\rho v^3 AC_D \qquad P_A = vma$$

Resulting in:

$$P_{total} = \frac{v(mgC_{rr} + \frac{1}{2}\rho v^2 AC_D + ma)}{\eta}$$

Where v is velocity, m is mass, g is acceleration due to gravity, C_{rr} is coefficient of rolling resistance. ρ is air density A is vehicle frontal area, C_D is vehicle drag coefficient, and a is acceleration. All units are SI units. The power it takes to maintain any velocity v (set $a = 0$) can be solved for, given that the vehicle,environment, and driver properties C_D, C_{rr}, m, g,and ρ are known. See Appendix A - *Solving the Power Equation* for an example. In this application the power equation is used to solve for acceleration a at discretized time steps where current velocity is known. See Appendix A - *Acceleration Rearrangement* for the rearranged acceleration equation.

In addition to calculating the accumulation of velocity over time, a fatigue model has been built to approximate the available energy a driver has remaining to propel the vehicle. Fatigue F accumulates as a function of power p and duration of that power input t, according to the following equation:

$$F = \frac{t}{T_{max}(p)}$$

A T_{max} curve represents a relationship between power, and the maximum duration a driver can maintain that power. An example curve can be found in Appendix A - *Fatigue Curve*. The fatigue model assumes constant accumulation of fatigue. If a driver can maintain a 400 watt input for 100 seconds and reach full fatigue $F = 100\%$, 10 seconds of 400 watt input accumulates $F = 10\%$ or 10% fatigue.

Finally, there is the calculation of Coefficient of rolling resistance C_{rr}. In this case, it is assumed to be solely a function of velocity v and can be found via the function reproduced in Appendix A - *Rolling Resistance*.

3 Q-Learning Implementation

Q-Learning is a reinforcement learning algorithm used to assign expected values to state-action pairs. A Q-learned agent always selects the action that maximizes expected value at every state. This final set of state-action selections is called a policy. The goal in this application of Q-learning is to generate a driver, environment, and vehicle specific policy starting at 0 velocity v and 0 fatigue F. The learned policy should maximize velocity while not exceeding a $F = 100\%$ fatigue threshold. for specifics about the Q-learning algorithm see Appendix B - *Q-Learning Algorithm*. The primary features of Q-learning in this application are the state space, action space, and environment.

3.1 State, Action, Environment

A **state** is comprised of a velocity v and fatigue F pair. This ensures that all relevant information needed to calculate a reward ($Reward = \frac{velocity}{fatigue+1}$) is encoded. Further, this simple state definition keeps the problem size manageable.

The **action** space is any power p to be input by the driver for the duration of the next time step. The state and action space must be discretized in such a way to balance accuracy of the simulation, while maintaining a tractable problem size.

The **environment** is the set of factors that govern the simulation results, and are constant throughout the simulation runs. The driver has cycling capability, and weight. The vehicle has weight, frontal area, and drag coefficient. And the external environment has gravity and air density. These are all components of the simulation environment.

3.2 Training the Agent

With a selection of environment variables, simulations are run to find an optimal policy given the environment. Each simulation starts at state space ($0v$, $0F$) where v is velocity and F is fatigue. The agent chooses an action based on two factors the hyper-parameter ϵ which determines how often the agent takes a random action, and the q-value of the available state-action pairs. If the agent does not act randomly, it chooses the action with the maximum q-value. From here the simulation follows these steps:

- If the current state fatigue F is equal to or greater than 1, the simulation stops.
- Given the action, the next velocity state and fatigue are calculated, see appendix B - *Next Velocity, Next Fatigue*.
- The Environment steps t_s seconds in time.
- Reward is calculated and the q-value of the previous state is updated.
- The current state (v_0,F_0) becomes the next state (v_1,F_1)

Q-values are maintained between simulations, each simulation continues to update Q-values.

4 Results

Results from applying Q-Learning to maximize velocity and minimize driver fatigue in a human powered vehicle are explored below. The environment, Q-learning hyper-parameters, and simulation setup are found in Appendix C - *Setup*.

250 simulations were run, where the agent gradually increases velocity and fatigue from starting state ($0v$,$0F$) to some state (v_n,$1F$), after n time-steps by selecting an action at each step. The total duration of each simulation was not constant due to the variability in fatigue accumulation by way of action selection. The top 10 fastest runs actions over time are reproduced below: All

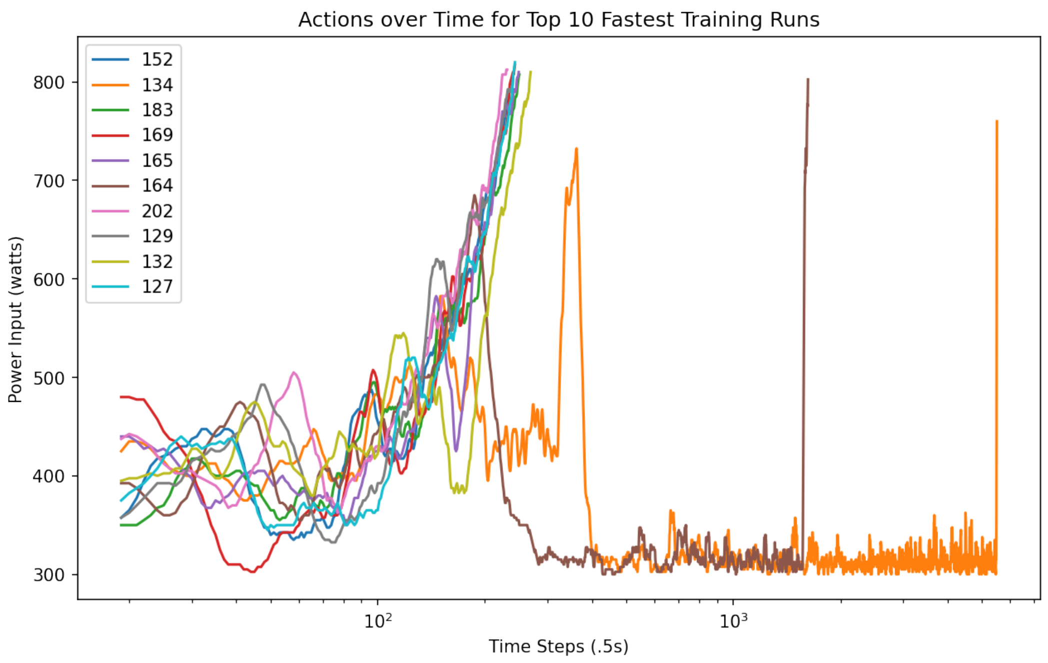

Figure 1: Top 10 training runs 10 second mean smoothed actions over time

simulations increase power over time and incur fatigue for doing so. The top 10 simulations follow similar paths increasing power gradually to maximize velocity while minimizing fatigue. This behavior is attributed to the reward function that incentivizes velocity gain for minimal fatigue increase at every time step. As velocity increases, the power required to continue to accelerate increases, and the fatigue accumulates faster. Discussion about the orange and brown simulations visualized above follows in appendix C - *Discretization and Time Step*.

Ultimately, highly optimized power-input over time functions are achievable via the Q-Learning process. See the result of a 250 iteration Q-learned agent below. The learned policy resulted in a maximum velocity of 70.4 miles per hour. The Q-learned agent outperformed constant actions over time, ramp functions over time, and outperformed human selected actions over time consistently. See table below figure 2. Due to the cubic relationship between power to overcome aerodynamic drag and velocity, increasing the maximum speed of the vehicle just one or two miles per hour could require herculean power inputs.

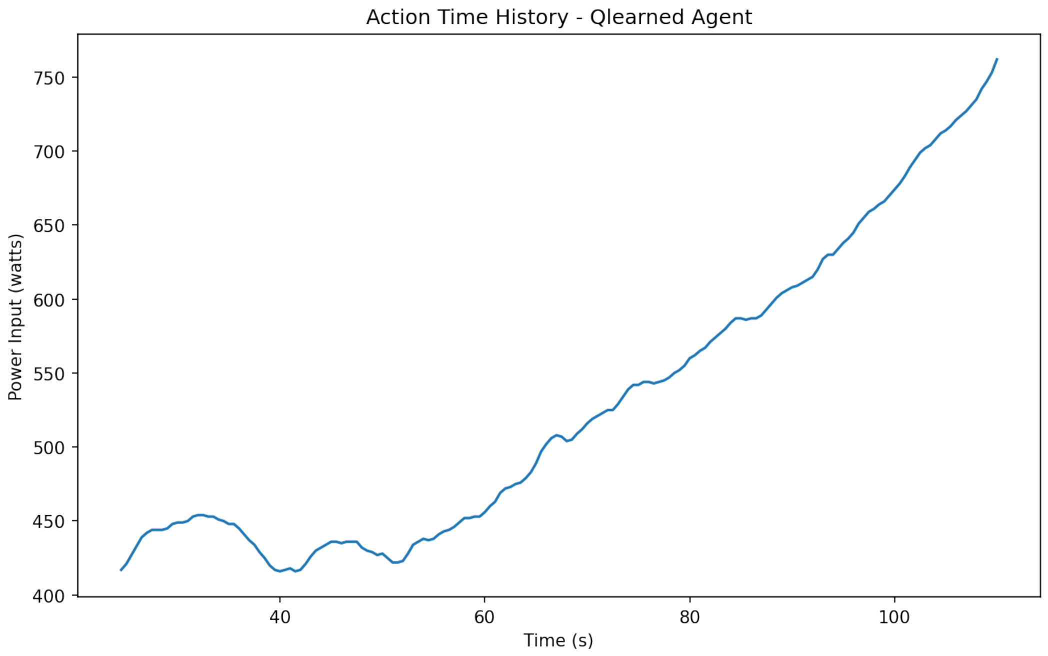

Figure 2: Rolling 25 second smoothed Q-learned agent actions over time

Agent	Max Velocity Achieved (mph)
Constant 800w	46.2
Constant 700w	48.0
Constant 300w	55.6
Constant 500w	58.9
Constant 450w	60.8
Constant 375w	61.8
Constant 400w	62.0
Ramp 0-500w ($a = \frac{.5watt}{second}$)	65.6
Ramp 0-700w ($a = \frac{1watt}{second}$)	65.9
Human (manual)	67.6
Q-Agent	70.4

5 Discussion

Q-learning appears to be a useful tool for gauging speed achievable in a human powered vehicle. With the ability to set driver, vehicle, and environment variables for the conditions, the tool is flexible in its application. A necessary next step would be to validate the fatigue model, and test the power profiles to determine if a rider of a given weight and capability can perform the power input over duration policy. Further, making the problem larger (by discretizing fatigue,velocity, and power with smaller steps) and running more simulations might provide even faster runs, or at least conclude smoother more more reasonable actions over time. One extension of this work is to optimize multi-driver vehicle configurations, the action space would then include multiple drivers power inputs over time, this would increase the problem size, and would require more simulations to find a "best" solution. The Q-Learning process could explore very interesting combinations of power by 2 or more drivers.

References

[1] Wilson, D. G., &amp; Schmidt, T. (2020). Bicycling science (4th ed.). Cambridge, MA: MIT press.

[2] Johnstone, D. (2018, June 7). How does your cycling power output compare? Retrieved 2021, from https://www.cyclinganalytics.com/blog/2018/06/how-does-your-cycling-power-output-compare

[3] Morrison, A. (2010, March 20). AFM tire testing rev9. Retrieved 2021, from http://www.biketechreview.com/tires_old/images/AFM_tire_testing_rev9.pdf

6 Appendix A - Governing Physics

6.1 Solving the power equation

An example set of variable values:

Mass	m	$100kg$
Gravitational Acceleration	g	$9.81\frac{m}{s^2}$
Air Density	ρ	$1.25\frac{kg}{m^3}$
Vehicle Frontal Area	A	$0.4m^2$
Drag Coefficient	C_D	0.1
Drivetrain Efficiency	η	0.97
Acceleration	a	$0\frac{m}{s^2}$
velocity	v	$17\frac{m}{s}$
Coefficient of Rolling Resistance	C_{rr}	0.007

Plugging the variables in:

$$P_{total} = \frac{17(100 * 9.81 * .007 + \frac{1}{2} * 1.2517^2 * .4 * .1 + 100 * 0)}{.97}$$

$$P_{total} \approx 240watts$$

6.2 Acceleration Rearrangement

Rearranging the power equation for acceleration:

$$a = \frac{vmg * C_{rr} + \frac{1}{2}\rho v^3 AC_D - \eta P_{total}}{-vm}$$

6.3 Fatigue Curve

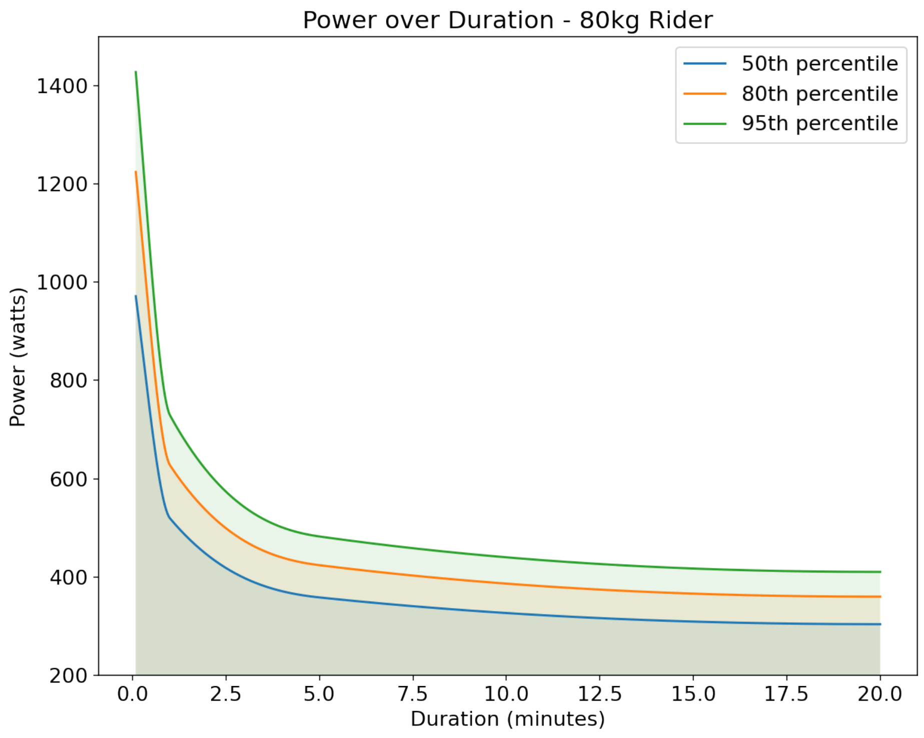

Figure 3: 80kg driver fatigue curves

6.4 Rolling Resistance

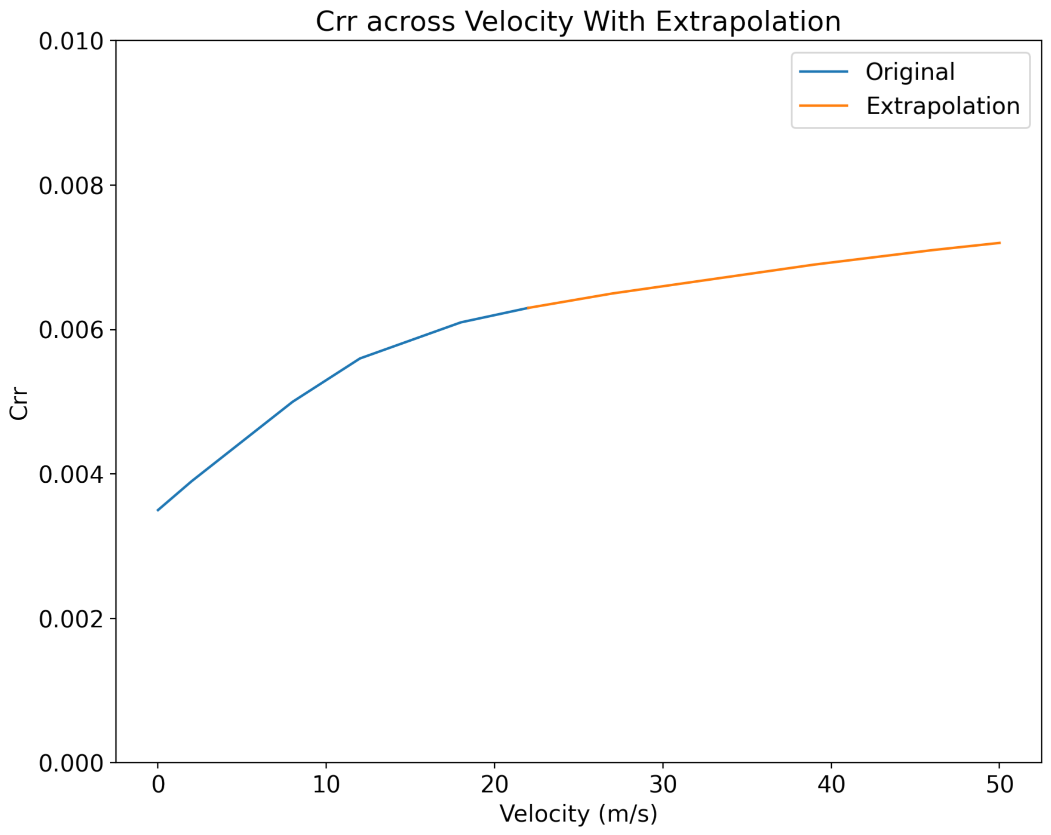

Figure 4: C_{rr} as a function of velocity

7 Appendix B - Q-Learning

7.1 Q-Learning Algorithm

The Q-Learning algorithm is implemented exactly as prescribed in literature.

- Given a current state, take an action according to randomness or maximum Q-value
- Calculate reward and discount times maximum Q-value of next state-action pairs.
- Update Q-value of current state action pair

The Q-Learning update incorporated is as follows:

$$Q_{n+1}(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha[Reward(s,a,s') + \gamma \max_{a`} Q_n(s,a)]$$

7.2 Next Velocity, Next Fatigue

Next velocity is the velocity of the vehicle at the next time step after taking an action at the current time step. It is calculated using the power equation and is implemented in code here:

```
def next_velocity(time_delta, current_velocity,
              power_in, vehicle, density=1.07):
    PR=Pr(v=current_velocity, m=vehicle.mass,
          Cr=Crr(current_velocity), eta=vehicle.eta)
    PD=Pd(p=density,v=current_velocity,A=vehicle.A,
          Cd=vehicle.CD, eta=vehicle.eta)
    PA = (power_in - PR - PD)*vehicle.eta
    A = PA/(vehicle.mass*current_velocity)
    NV = current_velocity+A*time_delta
    return NV

def Pd(p,v,A,Cd,eta=.96):
    '''Calculate power to overcome aerodynamic drag.'''
    return .5*p*v**3*A*Cd/eta

def Pr(v,m,Cr,eta=.96):
    '''Calculate power to overcome rolling resistance'''
    return v*m*9.81*Cr/eta

def Pa(v,m,a,eta=.96):
    '''Calculate power to accelerate.'''
    return v*m*a/eta

crr = [.0035,.0039,.005,.0056,.0061,.0063,.0065,.0067,.0069,.0071,.0072]
vs = [0,2,8,12,18,22,27,33,39,46,50]

crr_df = pd.DataFrame(crr, index=vs, columns=['Crr'])
crr_df = crr_df.reindex(np.arange(0,50.01,.01)).interpolate('pchip')
Crr = interp1d(crr_df.index, crr_df['Crr'])
```

Next fatigue is calculated by summing current fatigue with the accumulation of fatigue over the next time step given a power input. Next fatigue is implemented in code here:

```
def fatigue(action,t0,t1,fatigue_curve):

    ts=t1-t0

    return ts/fatigue_curve(action)
```

Note that the fatigue curve is rider dependent. Some are shown in Appendix A - *Fatigue Curve*

8 Appendix C - Results

8.1 Setup

Alpha (learning rate)	α	.5
Gamma (discount)	γ	.1
Epsilon (random action rate)	ϵ	.25
Number of Simulations	n	250
Time Step	t_s	$.5s$
driver Mass	m_r	$80kg$
driver Percentile	R_p	80%
Vehicle Mass	m	$20kg$
Gravitational Acceleration	g	$9.81\frac{m}{s^2}$
Air Density	ρ	$1.07\frac{kg}{m^3}$
Vehicle Frontal Area	A	$0.325m^2$
Drag Coefficient	C_D	0.0496
Drivetrain Efficiency	η	0.97
Starting Acceleration	a	$0\frac{m}{s^2}$
Starting Velocity	v	$0\frac{m}{s}$
Starting Fatigue	F	$0\frac{m}{s}$
Coefficient of Rolling Resistance	C_{rr}	See Fig 4.

8.2 Discretization and Time Step

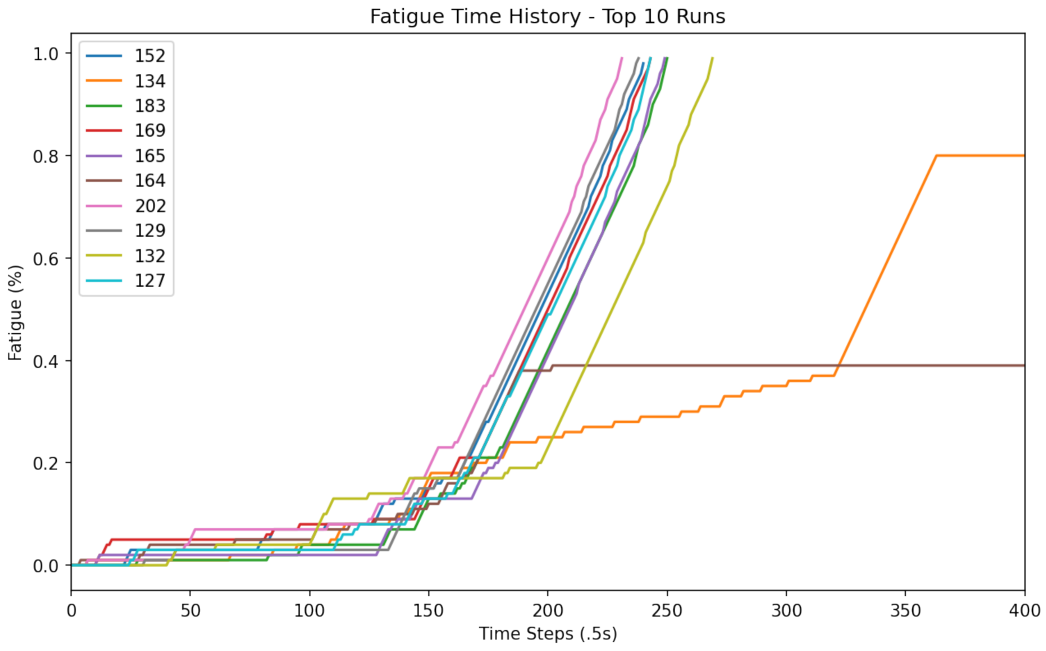

Figure 5: Top 10 learning runs fatigue over time

Notice in figure 5 (above) and 6 (on the next page), the orange and brown simulations resort to around 350 watt power inputs for an extended duration while fatigue remains constant. This is a result of time step and discretization selections. Given a small enough time step, and a broad enough fatigue discretization, the accumulation of fatigue over the course of the time-step is insufficient to round to the next highest fatigue level. This results in fatigue remaining constant over the time step, which is very rewarding to the agent at high enough velocities. Eventually the agent reaches a point where the given power neither increases fatigue or velocity enough to move up one discretized level. The agent is then incentivized to increase power. One method to mitigate this issue, is to discretize fatigue into more states, at the expense of increased problem size. An alternative mitigation, is to

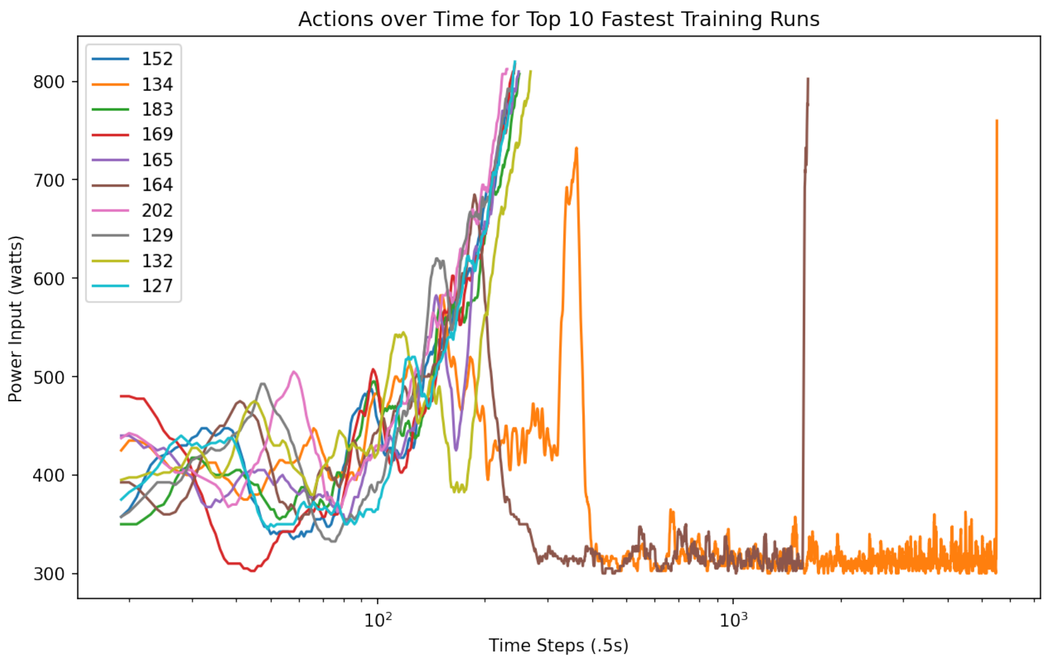

Figure 6: 10 second mean smoothed actions over time

change reward from:

$$Reward = \frac{velocity}{fatigue + 1}$$

to:

$$Reward = \frac{acceleration}{fatigue + 1}$$

Incentivizing acceleration ensures that the the agent seeks to continually increase speed, and penalizes constant velocity, unlike the prior reward function. With this reward adjustment, the top 10 learning runs can be found on the next page.

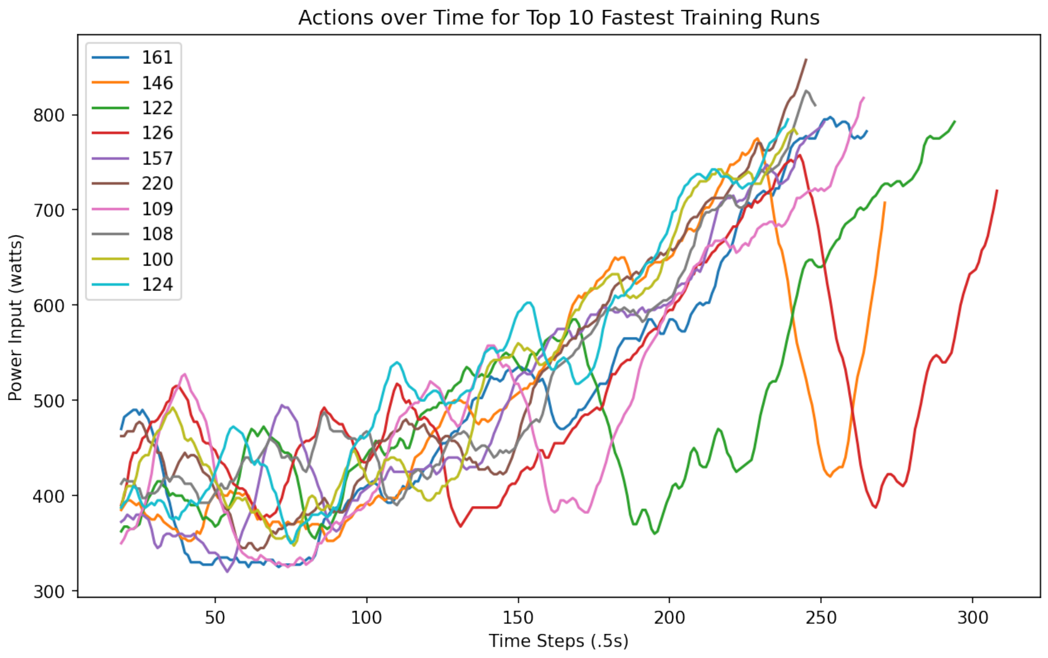

Figure 7: 10 second mean smoothed actions with acceleration reward function