EAEEE 4000 - Final Project

Emissions Reductions Planning: Learning an Optimal Policy with Reinforcement Learning

Melanie Subbiah (mss2290)

December 24, 2021

Abstract

Global warming is one of the critical problems facing the current generation. If we do not take action now to limit the effects of global warming, the consequences could be disastrous in terms of natural disasters, and loss of livable habitats for humans and other species on the planet. There is a global goal to limit global warming to 1.5º C relative to pre-industrial levels (with a relaxed goal of 2º C), but this goal does not come with a clear roadmap of the best way to accomplish it. In this project, I use a simple climate simulator and try to apply reinforcement learning to the problem of determining an optimal emissions reduction strategy over the next 80 years to have warming under control by 2100.

1 Introduction

In 2015, 196 political entities from around the world agreed to the Paris Agreement. This international treaty was a pledge to curb global warming, such that temperatures did not rise more than 1.5º C relative to pre-industrial levels (par). As of this year the world has already warmed by about 1º C (see Figure 1), and we are on track to surpass the 1.5º C warming target by 2040, if emissions continue to grow accoring to the current pattern.

Global warming is caused by several different actions and interconnected systems (see Figure 3). Greenhouse gases that are emitted by humans, like carbon dioxide and methane, trap heat

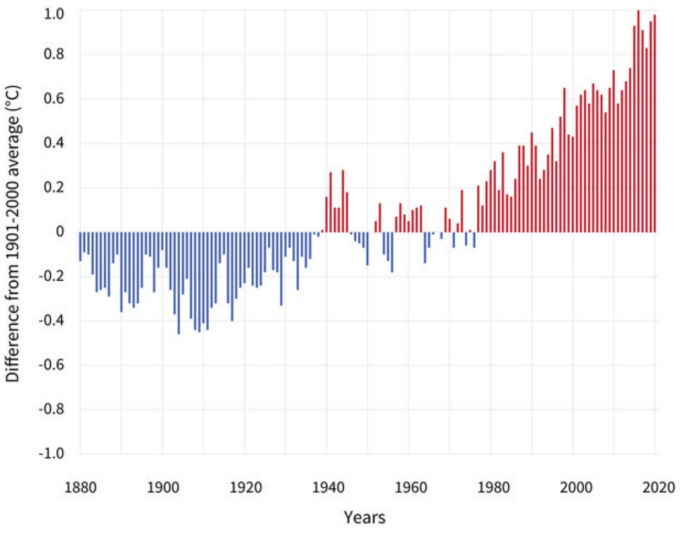

Figure 1: Global average surface temperature relative to pre-industrial levels (graphic from cli).

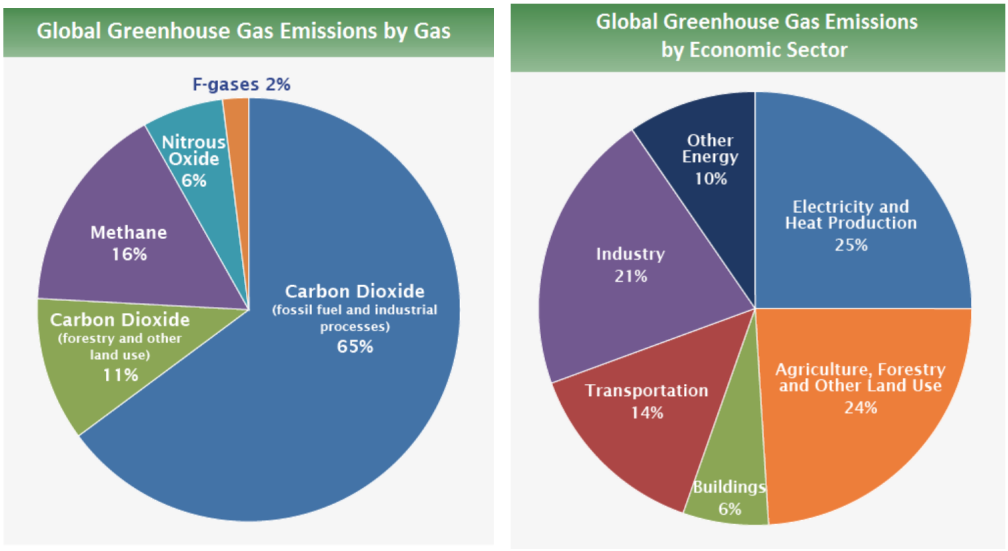

Figure 3: A breakdown of the different gases and industrial sectors that contribute to greenhouse gas emissions (graphic taken from epa).

in the atmosphere, hence the name "greenhouse". Some other radiative forcing factors are also naturally caused like volcanic eruption. If global warming exceeds the 1.5^{o} temperature goal, the consequences could be catastrophic around the world. With this limit in place, the world may experience more natural disasters and heatwaves (as we are already seeing), but the goal is to preserve the water supply and ecosystems necessary to sustain the diversity of life on this planet (nas).

While it is clear that we need to take action now to limit global warming, it is not clear what the right pathway to the goal is. For example, would it be better to reduce emissions as quickly as possible, or to take a more gradual approach? While the former might have a faster effect on the planet, the latter could be more affordable and realistic. Additionally, there are complicated aspects of the carbon cycle, such as how the ocean stores and releases carbon dioxide that are sensitive to the concentration of CO2 in the atmosphere (see Figure 4). Therefore, reducing emissions too quickly could mean that the ocean stops absorbing as much CO2 and we have to do more work to reduce the car-

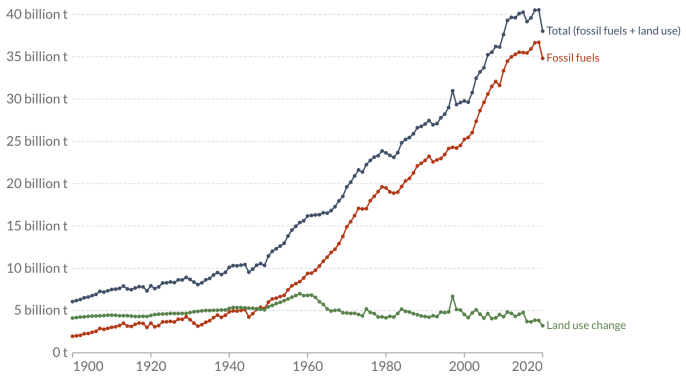

Figure 2: Global CO2 emissions in GtC, gigatonnes of carbon (graphic taken from owi). Fossil fuel emissions are from burning fossil fuels, whereas land use emissions are from repurposing land (i.e., deforestation releases carbon dioxide).

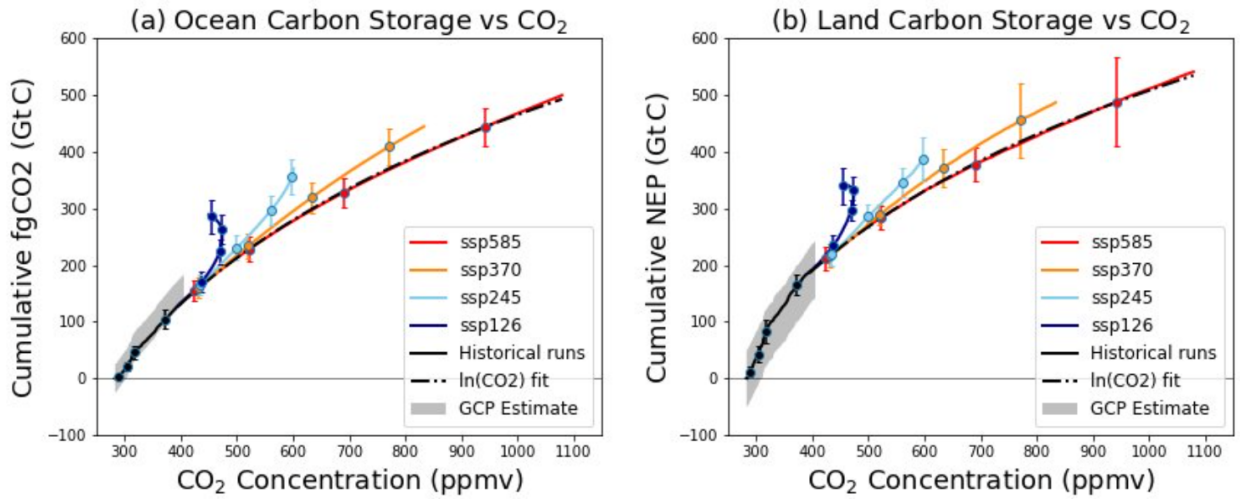

Figure 4: Modeling of how the land and ocean act as carbon sinks relative to the concentration of CO2 in the atmosphere (graphic taken from Peter Cox, cox).

bon dioxide concentration ourselves.

Previous work has focused on researching realistic possible future emissions scenarios and simulating how these might play out, but I have not seen as much work on planning an optimal emissions reduction policy taking into consideration climate effects and real-world constraints and costs. Framing the problem as a planning problem makes it an ideal candidate for reinforcement learning. Reinforcement learning is a type of machine learning that allows an agent to learn optimal actions to take in an environment to achieve some goal. The agent learns through taking actions in the environment and receiving feedback in the form of a reward (which can be positive or negative).

In this project, I apply reinforcement learning to the problem of optimal emissions reductions. Framing the problem with the Paris Agreement, I will focus on the next 80 years (until 2100) as my timeline, with the goal that warming is consistently under 1.5º C by then. Given that carbon dioxide is the most significant greenhouse gas, I will also focus my analysis purely on this gas for simplicity, but future work could consider modulating emissions across multiple gases.

In this report, I will first introduce the key technologies I used - the FaIR simulator, and reinforcement learning. I will then walk through the setup of my experiments and the results I observed. Finally, I will conclude with a discussion of the results and next steps for this line of work.

2 Background

2.1 Reinforcement Learning

Reinforcement learning (RL) is a method to train a model to handle sequential decision making tasks. It has become popular in recent years as advances in computing resources and efficiency have enabled combining deep learning with reinforcement learning successfully.

Researchers are exploring applications of RL across areas like robotics, self-driving cars, and advanced game play. The system that famously beat the best Go player in the world involved reinforcement learning. There have been fewer applications of RL within environmental science and sustainability to date, which is part of what motivated my interest in this project. Much of the information in this section is drawn from Levine et al.'s tutorial and survey paper (Levine et al. [2020]).

Reinforcement learning necessitates understanding some key terms:

- The *state* is the condition of the environment. The state can be fully-observed, meaning the model has access to all information about the state, or partially observed meaning some information is unknown to the model. State can be computationally represented in many ways, but typically it is a vector of values or an image.
- An *action* is a way that the model can interact with the environment. You have to define the action space - the range and bounds of actions the model can take. Different algorithms accommodate continuous vs. discrete action spaces.
- A *trajectory* is a sequence of states and actions in order. Sometimes this is also called a rollout.
- A *policy* (π) is the function the model uses to map states to actions. Often this is represented as a probability distribution over actions, but there are many different ways to represent the policy. With deep reinforcement learning, the policy is often parameterized as a neural network.
- The *reward* is the feedback from the environment on whether the model's actions are good or bad. Positive signals tell the model that it is doing something well. The reward signal guides the model toward the task goal.

With reinforcement learning, there are typically two loops within the learning process (see Figure 5). The inner loop is a cycle of the model taking actions in the environment based off of the current state, and then receiving back an updated environment state and reward based off of the action it chose. The outer learning loop then collects these trajectories and updates the model to a better policy. The learning process switches back and forth between these two loops, updating the policy and then collecting more data using that improved policy.

The general goal with the training process is to maximize the following learning objective:

$$J(\pi) = E_{\tau \sim p_\pi(\tau)} \left[\sum_{t=0}^{H} \gamma^t r(s_t, a_t) \right] \tag{1}$$

Essentially, this objective looks at the expectation across the probability distribution of trajectories defined by the learned policy and takes a discounted summation of rewards across each timestep in the sequences. The discount factor gives lower weight to rewards further in the future of the trajectory since they have more inherent uncertainty.

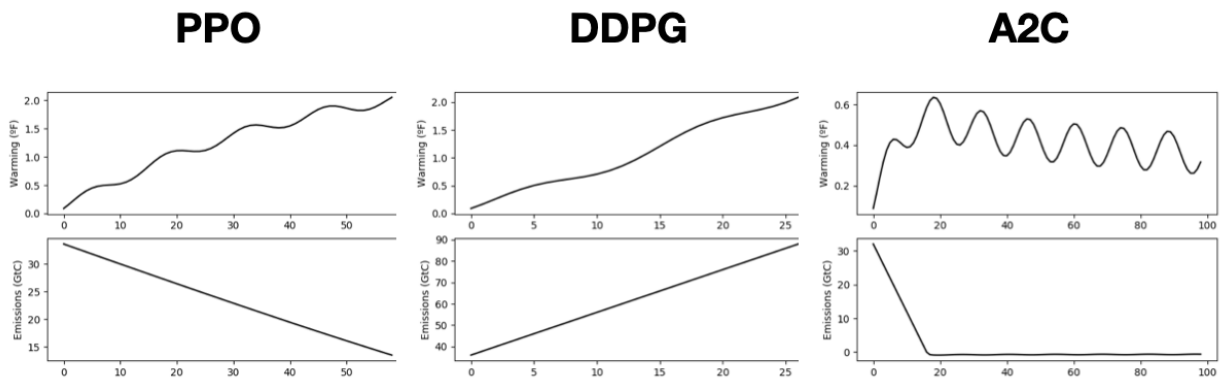

Figure 6: Results from preliminary experiments with different RL algorithms.

RL algorithms fall broadly into four families. Policy gradient methods learn the policy directly, and often parameterize the policy as a neural network. Approximate dynamic programming approaches learn the value function over states and actions. The policy then just becomes choosing the action which maximizes the value function for any given state. Actor-critic approaches learn both the policy and value function, the idea being that the learned value function can give better feedback to the policy during training. Finally, model-based approaches incorporate real-world knowledge of transition dynamics between states given actions.

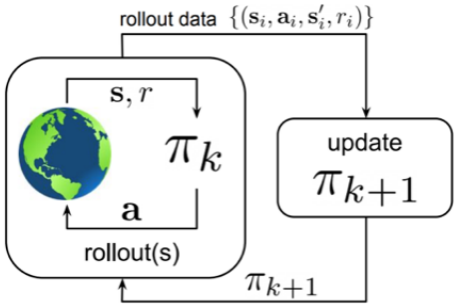

Figure 5: The typical learning loops involved in reinforcement learning (graphic taken from Levine et al. [2020]).

For this work, I wanted to try a couple different RL algorithms to see which would work well before focusing in on one approach. I experimented with Proximal Policy Optimization (PPO) (Schulman et al. [2017]), Deep Deterministic Policy Gradients (DDPG) (Silver et al. [2014], Lillicrap et al. [2015]), and Advantage Actor Critic (A2C) (Mnih et al. [2016]). All three of these are commonly used algorithms in the RL literature. PPO is a policy gradient approach which incorporates a novel penalty for large updates to the policy to try and stabilize learning. DDPG combines policy gradients and value function learning to make an actor-critic approach that can handle continuous action spaces. Finally, A2C is an actor-critic approach which is a synchronous version of A3C. For brevity, I will only elaborate on the algorithm I ended up using, and will do so in Section 3.2.

I likely could have tuned each algorithm better for my use case, but running with default parameters, I saw that A2C was a clear winner for my use case. I gave the models a simple reward that was positive if the temperature decreased in a given timestep and was largely negative if the temperature exceeded 2° C, meaning a failure. Each trial was run until failure or up to 100 years. Emissions were started at 36 GtC, and radiative forcing was a sinusoidal function, hence the waving in the temperature results.

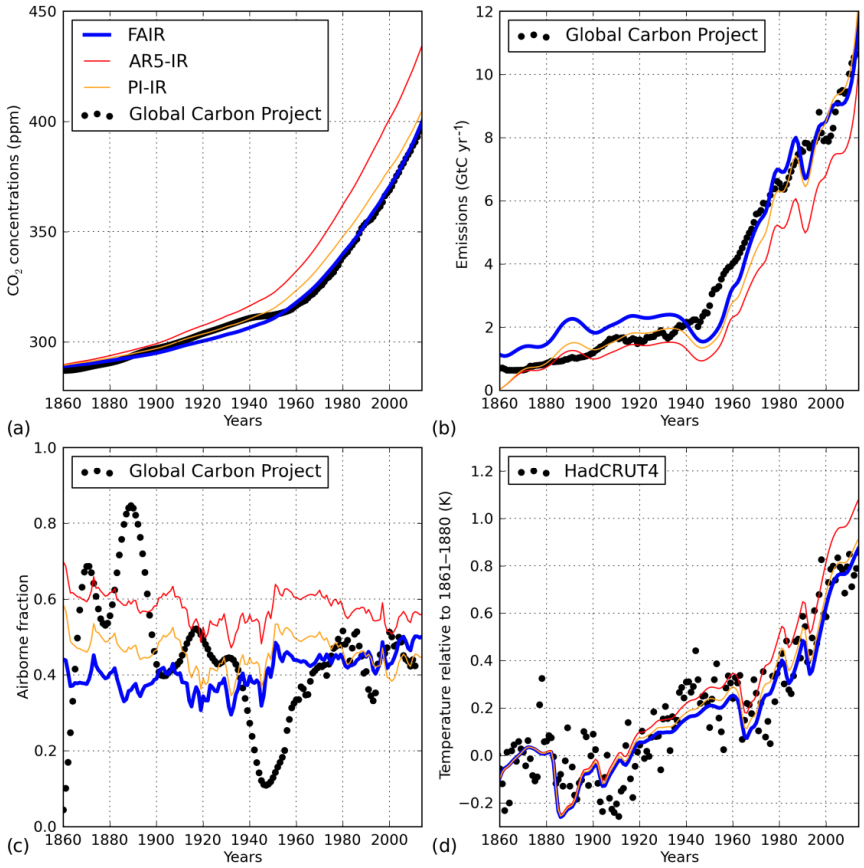

Figure 7: A comparison of the FaIR model prediction (in blue) against historical data (in black) and other existing models (graphic from Millar et al. [2017]).

Figure 6 clearly shows that A2C learned best under this regime. PPO learned to reduce emissions, but not fast enough to keep the temeprature under control. DDPG did not learn to reduce emissions at all. And A2C learned to reduce emissions fast enough to keep the temperature under control. Based on these results, I moved forward with A2C for the remainder of the experiments.

2.2 FaIR Climate Model

As we learned from the previous section, having an environment to interact with is a very important part of reinforcement learning. Obviously, we can not train an emissions planner with the real world since the feedback cycle would be extremely slow and actions the model took with emissions could have detrimental effects on the globe. Therefore, we have to use some sort of simulator.

Climate scientists have been studying how to model the carbon cycle and climate effects for a long time and several different simulators exist. For this project, I chose to work with the Finite Amplitude Impulse Response (FaIR) simulator because it can run a simulation of hundreds of years in under a second, and it is written in Python so it plugs in easily with Python-based deep learning libraries (Millar et al. [2017], Smith et al. [2018], fai). See Figure

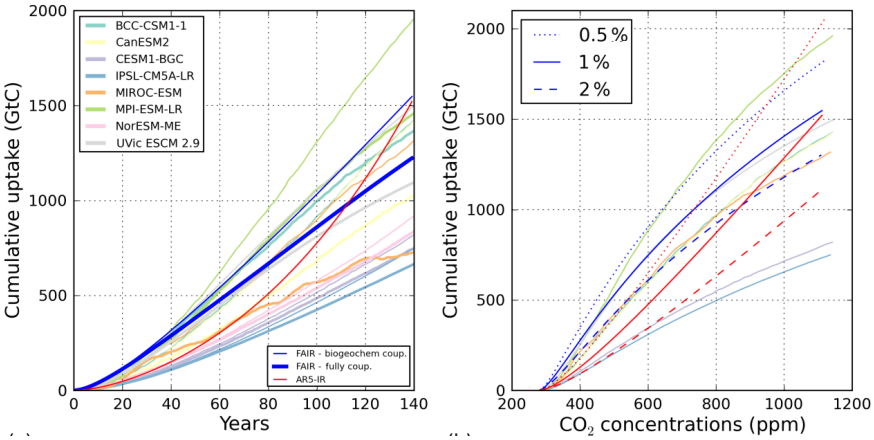

Figure 8: The model's cumulative land and ocean carbon uptake over time under different regimes of % C02 concentration increase per year. The FaIR predictions are in blue (graphic from Millar et al. [2017]).

7 for a comparison of this model against other models and against real data.

FaIR is an emissions-based model which accepts a vector of emissions over a period of years as input. The model then computes and outputs the corresponding greenhouse gas (GHG) concentrations, radiative forcing, and temperature anomaly at each timestep. Temperature anomaly means the temperature difference relative to pre-industrial levels. The model computes non-CO2 radiative forcing as a function of the emissions, with the exception of solar and volcanic forcing since these are natural phenomenons and therefore must be input separately. Given this setup, the model is ideal for assessing how future emissions sequences may play out in terms of GHG concentrations and temperature.

There are several different modes to run the simulation in. Most notably, it can be run in carbon-only or multigas mode. In carbon-only mode, it expects an nx1 vector of C02 emissions where n is the number of years. In multigas mode, it expects an nx40 emissions vector. The 40 inputs per year are the year itself, then CO2 emissions from fossil fuels, CO2 emissions from landuse change, and then emissions of 37 other GHGs.

The model outputs temperature anomaly as an nx1 vector, concentrations of GHGs in the atmosphere, and radiative forcing. The concentrations will either be an nx1 vector of just CO2 concentrations in carbon-only mode, or an nx30 vector in multigas mode. There are only 30 concentrations output because some of the gases had multiple emissions specified in the input. Finally, the radiative forcing is an nx1 vector of total forcing in carbon-only mode, or an nx13 vector in multigas-mode. The 13 forcing outputs are CO2, CH4, N2O, all-other well-mixed GHGs, tropospheric O3, stratospheric O3, stratospheric water vapor from CH4 oxidation, contrails, aerosols, black carbon on snow, land use change, volcanic, and solar.

FaIR sets default parameters for the carbon cycle, but users can modify these parameters so as to measure uncertainty of how scenarios would change based on uncertainty in modeling parameters. There are input parameters to control rates around re-absorption, equilibaration, and invasion in land and carbon sinks. In particular, $r0$, rc, and rt control the pre-industrial sensitivity of carbon sinks, the sensitivity to cumulative carbon dioxide

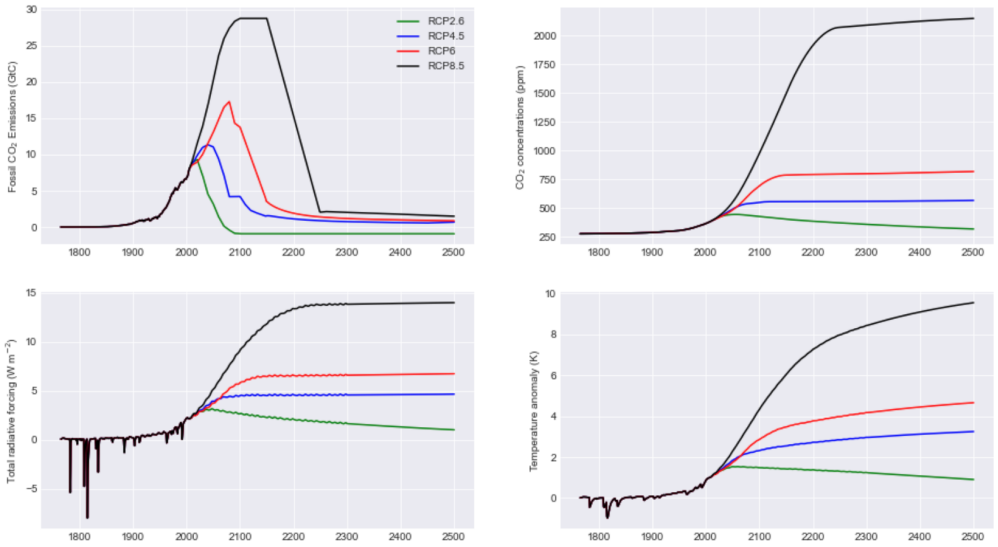

Figure 9: How different RCP scenarios play out in the FaIR model (graphic from fai)

.

emissions, and the sensitivity to temperature change. See Figure 8 for an example of how total CO2 land and ocean uptake plays out in the model.

Finally, for ease of use, the model couples with some external data sources. Van Vuuren et al. developed a set of four Representative Concentration Pathways (RCPs) which have been studied as possible future pathways for CO2 concentrations (Van Vuuren et al. [2011]). RCP2.6 is the most mild whereas RCP8.5 follows a more extreme emissions scenario (see Figure 9). The emissions for these scenarios can be loaded in the FaIR model. I use these values to help initialize the model to the current world condition and provide emissions for other gases when running in multigas-mode.

3 Approach

3.1 Environment

The OpenAI Gym is a Python library built for running reinforcement learning experiments (gym). There are many environments built in to the library, but I had to create a custom environment to run my own experiments with it. For the most part, the environment I built just handles running the FaIR model using specified parameters. However, the critical pieces to define are the state representation and the reward function.

For the *state*, I simply used a vector representation. In carbon-only mode, the state is a 4x1 vector specifying the current temperature anomaly (^{o}C), CO2 emissions (GtC), CO2 concentration (ppm), and total forcing (Wm^{-2}). In multigas-mode, the state is similar but is a 16x1 vector to accommodate the specific forcing values. Once the temperature went above 2^{o}C warming, I stopped the episode as a failure.

For the action space, I allowed the model a continuous action space in the range $[-a, a]$, where a is some integer. An action is the model reducing or increasing CO2 emissions

relative to the previous timestep. So if the previous timestep emitted 36 GtC, and the current action was -2, then the current timestep would emit 34 GtC. I defined the actions this way rather than just having the model pick an emissions value because this approach creates more continuity between timesteps and ensures the model is reducing emissions at a rate that is feasible in the real world. Additionally, once emissions reached 0, I clipped any future actions so that the emissions would not become negative.

For the reward, I experimentd with many different functions:

- *simple* - This function rewards any temperature decrease relative to the previous timestep and strongly discourages the warming ever going above 2^{o}C. Here T_t indicates the temperature at timestep t and r_t indicates the reward at timestep t.

$$r_t = \begin{cases} T_{t-1} - T_t & T_t \leq 2 \\ -100 & T_t > 2 \end{cases}$$

- *temp* - This function rewards keeping the temperature under the 1.5^{o}C warming goal.

$$r_t = \begin{cases} 1.5 - T_t & T_t \leq 2 \\ -100 & T_t > 2 \end{cases}$$

- *conc* - This function rewards any decrease in CO2 atmospheric concentration. Here C_t indicates the CO2 atmospheric concentration at timestep t.

$$r_t = \begin{cases} C_{t-1} - C_t & T_t \leq 2 \\ -100 & T_t > 2 \end{cases}$$

- *carbon-cost* - This function imposes a negative cost for each GtC emitted. Here E_t indicates the CO2 emissions at timestep t.

$$r_t = \begin{cases} -E_t & T_t \leq 2 \\ -100 & T_t > 2 \end{cases}$$

- *temp-emit* - This function gives large positive and negative rewards for a sucessful or failed trajectory respectively. It also imposes a "cost" on emissions reductions with the goal of minimizing the amount of reduction necessary to still reduce temperature and meet temperature goals.

$$r_t = \begin{cases} -100 & T_t > 2 \\ 100 & t = 79, T_t \leq 1.5 \\ 10 * (T_{t-1} - T_t) - (E_{t-1} - E_t) & t < 79, E_t < E_{t-1} \\ 10 * (T_{t-1} - T_t) & t < 79, E_t \geq E_{t-1} \end{cases}$$

- *temp-emit-diff* - This function is similar to the previous one, but instead of taxing emission reductions from the previous timestep, it taxes reductions compared to projected

emissions for that year without reductions. This takes into a account the growing economic need for energy and resources. I estimated the projected emissions using the current slope of the emissions curve and the current emmissions.

$$r_t = \begin{cases} -100 & T_t > 2 \\ 100 & t = 79, T_t \leq 1.5 \\ 10 * (T_{t-1} - T_t) - ((0.6 * t + 36) - E_t) & t < 79, E_t < (0.6 * t + 36) \\ 10 * (T_{t-1} - T_t) & t < 79, E_t \geq (0.6 * t + 36) \end{cases}$$

3.2 Model

As mentioned previously, I used the A2C model for this project (Mnih et al. [2016]), as implemented in the OpenAI gym (gym). A2C was first introduced as A3C in 2016. A3C was an asynchronous actor-critic model. A2C applies the same approach but synchronously so that it can run more efficiently across multiple GPUs if necessary.

A2C maintains a policy $\pi(a_t|s_t;\theta)$ and a value function $V(s_t;\theta_V)$. In this implementation, both of these are parameterized as a multi-layer perceptron with shared weights, except the value function has a single linear output, and the policy has a softmax output. The algorithm uses backpropagation to update both the policy and the value function after n actions are taken by the policy.

The algorithm generally works to maximize the advantage function:

$$\sum_{i=0}^{k-1} \gamma^i r_{t+i} + \gamma^k V(S_{t+k;\theta_v}) - V(S_t;\theta_v) \tag{2}$$

where k varies by state but is capped by the maximum number of timesteps in a rollout. Intuitively, this function looks at the advantage of the current state over future states in the trajectory (i.e., is most of the state value coming from future states or is the current state critical?). A2C additionally adds the policy entropy to the loss to encourage exploration of different policies.

The full A3C algorithm from the paper is given in Figure 10. Removing the asynchronous components makes it the A2C algorithm.

3.3 Training Setup

I first ran some experiments with only 1000 timesteps to see which training hyperparameters seemed to work well. I used default settings of 'MlpPolicy' for the A2C policy, action space [-2, 2], *simple* reward, no radiative forcing, 14 random seed, learning rate of .0007, 5 steps between training updates, 0.99 gamma, carbon-only mode, and 'rcp60' scenario initialization. Gamma is the discount factor applied to future rewards and values. The scenario initialization uses emission values from the RCP scenarios to setup the last 250 years so that the model starts at the current temperature and CO2 concentration levels. I then tried some variants on different parameters as described in Table 1.

The mean episode reward is the average cumulative reward across the trajectories sampled from the model. A higher value is better for reward. The mean episode length is the average

Algorithm S3 Asynchronous advantage actor-critic - pseudocode for each actor-learner thread.

// Assume global shared parameter vectors θ *and* θ_v *and global shared counter* $T = 0$
// Assume thread-specific parameter vectors θ' *and* θ'_v
Initialize thread step counter $t \leftarrow 1$
repeat
  Reset gradients: $d\theta \leftarrow 0$ and $d\theta_v \leftarrow 0$.
  Synchronize thread-specific parameters $\theta' = \theta$ and $\theta'_v = \theta_v$
  $t_{start} = t$
  Get state s_t
  repeat
    Perform a_t according to policy $\pi(a_t|s_t;\theta')$
    Receive reward r_t and new state s_{t+1}
    $t \leftarrow t + 1$
    $T \leftarrow T + 1$
  until terminal s_t **or** $t - t_{start} == t_{max}$
  $R = \begin{cases} 0 & \text{for terminal } s_t \\ V(s_t, \theta'_v) & \text{for non-terminal } s_t \text{// Bootstrap from last state} \end{cases}$
  for $i \in \{t-1, \ldots, t_{start}\}$ **do**
    $R \leftarrow r_i + \gamma R$
    Accumulate gradients wrt θ': $d\theta \leftarrow d\theta + \nabla_{\theta'} \log \pi(a_i|s_i;\theta')(R - V(s_i;\theta'_v))$
    Accumulate gradients wrt θ'_v: $d\theta_v \leftarrow d\theta_v + \partial \left(R - V(s_i;\theta'_v)\right)^2 / \partial \theta'_v$
  end for
  Perform asynchronous update of θ using $d\theta$ and of θ_v using $d\theta_v$.
until $T > T_{max}$

Figure 10: Pseudocode for the A3C algorithm (pseudocode from Mnih et al. [2016])

.

Hyperparameter	Mean Episode Reward	Mean Episode Length	Policy Loss	Value Loss
(default configuration)	-53.5	49.9	2e-3	3e-6
Action space				
[-10,10]	-101	18.3	8e-2	3e-3
[-5,5]	-101	18.3	8e-2	3e-3
Learning rate				
0.01	-20.6	66.5	-4e4	2e-7
0.005	-96.4	22.9	4e-2	1e-3
Gamma				
0.95	-101	24.6	5e-4	9e-7
0.9	-101	20.8	2e-2	2e-4
N-steps				
1	-101	18.2	-3e-3	5e-6
10	-101	22.6	1e-1	1e-2

Table 1: Results from hyperparameter tuning.

Reward Mode	Mean Episode Reward	Mean Episode Length	Policy Loss	Value Loss
Reward Mode				
simple	-20.6	66.5	-4e4	2e-7
temp	-31.4	58.7	2e-2	5e-4
conc	-45.1	74.5	-2e-1	5e-2
carbon-cost	-20.6	66.5	-4e4	2e-7
temp-emit	-203	17	-12	63
temp-emit-diff	-110	17	7e-2	4e-3
With Forcing	-32.4	58.8	-7e-4	6e-6
RCP Scenario				
rcp26	-20.9	67.5	-5e-3	4e-4
rcp45	-31.6	59.5	-2e-2	1e-4
rcp60	-20.6	66.5	-4e4	2e-7
rcp85	-3.5	76.2	-3e-3	2e-6

Table 2: Results from experiments with different settings.

length of a trajectory. Since I stopped trajectories once the temperature got too high, a longer trajectory is better because it means the model is successfully keeping the temperature under control closer to the 2100 goal. The policy loss and value loss are the losses from training the policy and value functions with gradient descent. Lower values are better for losses. Based on these results, it seemed like the learning rate was the only adjustment that made a large positive difference in the results. I therefore used a .01 learning rate with the experiments going forward.

4 Results

For my experiments, I used different training settings. In all cases, I used the default parameters with the updated learning rate described in the previous section, other than the paramater I describe varying. I also trained the models for 5000 timesteps this time. I experimented with the different reward functions, using radiative forcing, and using the different RCP initialization situations (see Table 2). 'With forcing' means including initial proxies for solar and volcanic forcing.

As we can see from Table 2, the basic rewards (the first three) are learning quite well and achieving long episode lengths. However, the more complicated last two rewards that tradeoff between different priorities are not learning any useful policy. You can see this from their short episode length. This suggests that I need to do more work on reward shaping to properly guide the model through the tradeoffs being considered. The different RCP scenarios and the situation with forcing all seem to be learning well as well.

A full set of plots can be found in the 'outputs' directory of the code uploaded in conjunction with this report. However, I will walk through an analysis of useful associated plots using the example of the *conc* reward run.

For each training run, I used the policy deterministically (taking the max action instead

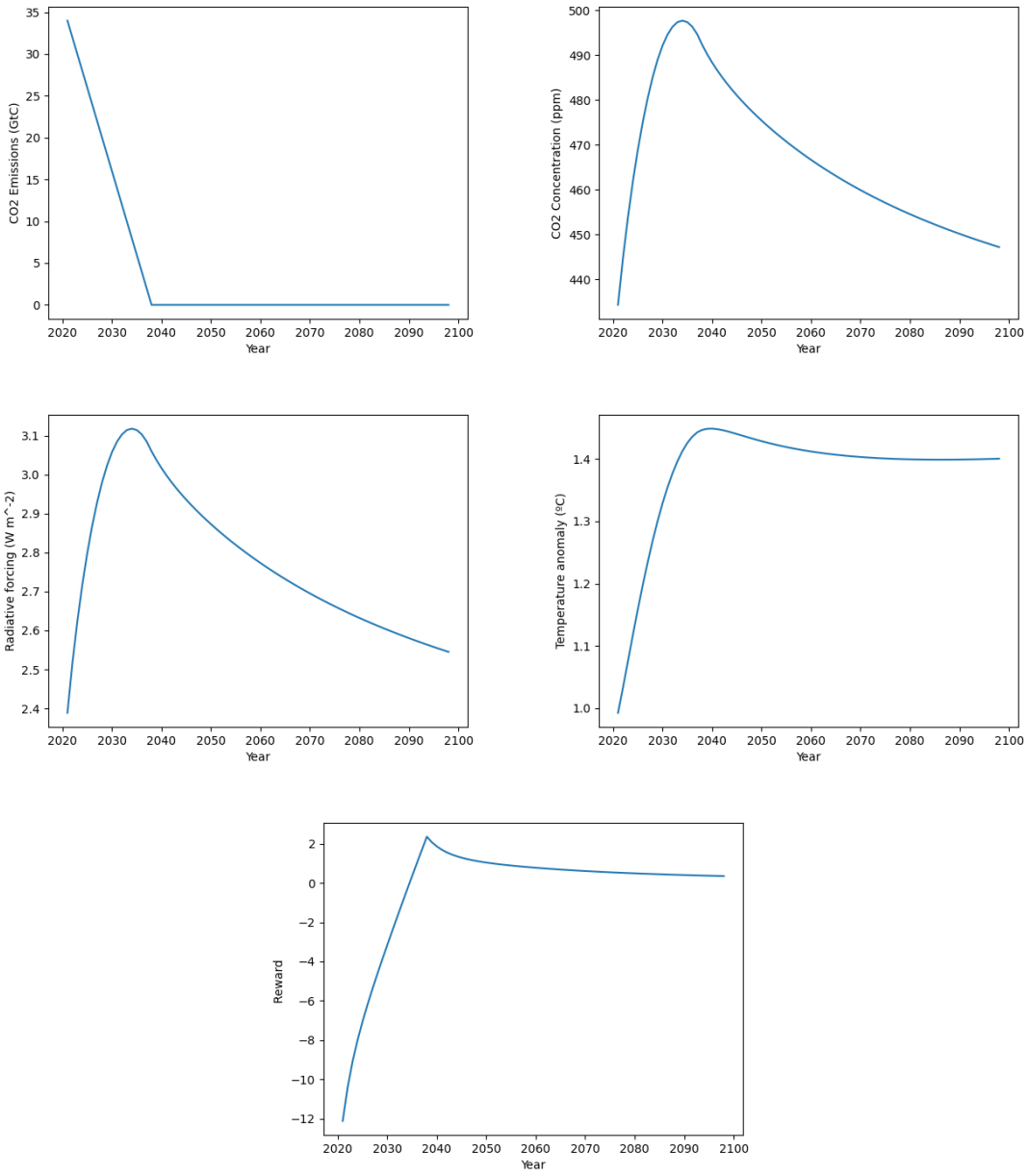

Figure 11: The Emissions reduction policy and associated effects learned with the 'conc' reward function.

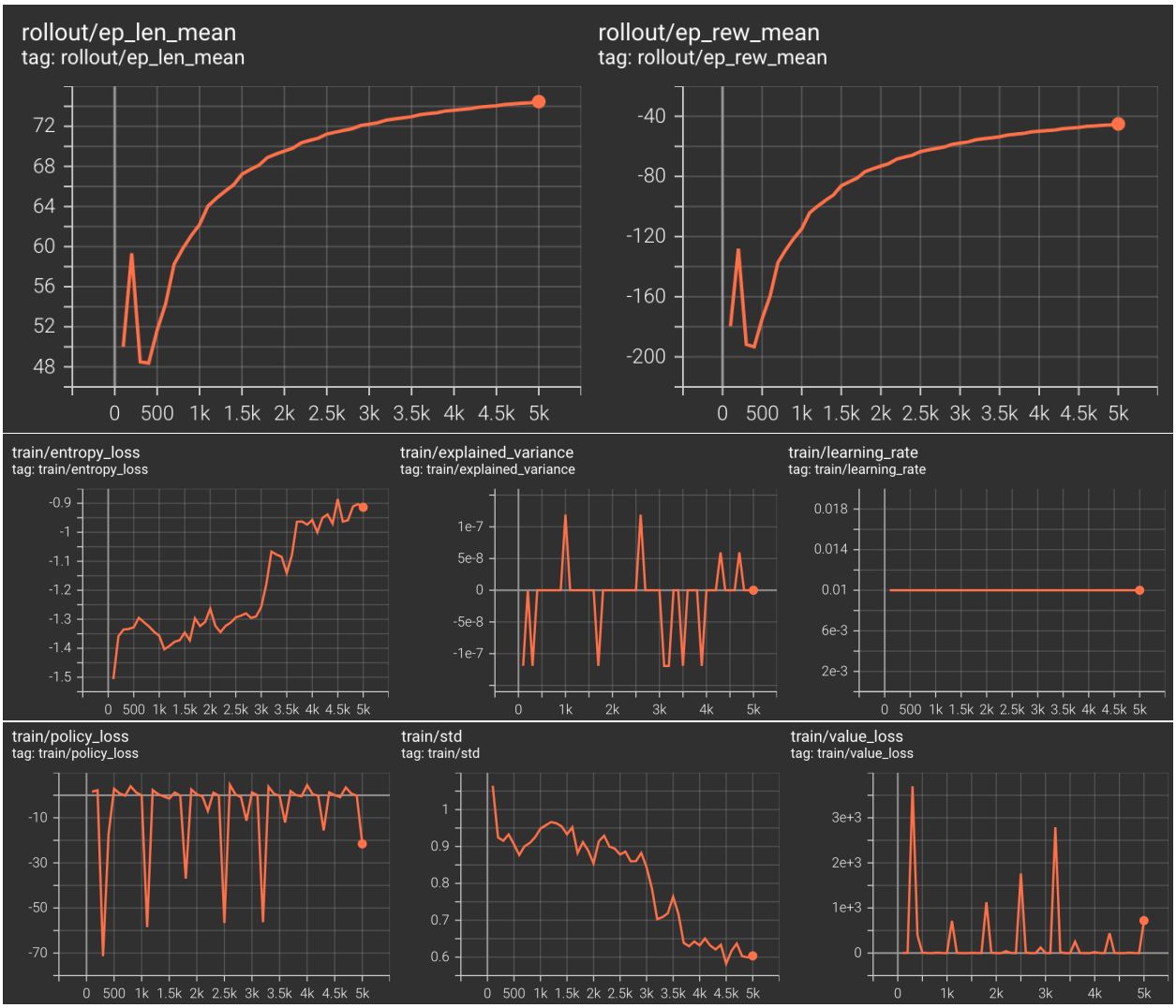

Figure 12: Training metrics tracked in Tensorboard for the ‘conc’ reward function run.

of sampling from the action distribution) after training to rollout the learned trajectory from the model. I then graphed the planned emissions, the associated outputs from the FaIR simulator, and the reward for the run. An example of these plots is shown in Figure 11. You can see the model learns to reduce emissions as quickly as possible with the *conc* reward function. The temperature therefore reduces quite quickly and remains under the 1.5^{o}C target. I am only showing one reward function example here because all three of the first rewards learned the same policy - reduce emissions as quickly as possible, which makes sense given how these rewards were formulated. In the future, I hope to get the more complicated tradeoffs working as well, but currently they just keep increasing CO2 emissions and fail quickly.

The OpenAI gym also logs useful metrics to Tensorboard during training. You can see an example of this output in Figure 12. The most easily interpretable metrics are the episode length and the episode reward shown at the top. Youc an see that both of these increase steadily throughout training which is good. However, the variability of the losses in the lower plot indicate that there is likely some instability in training.

5 Conclusion

Overall, this work represents a preliminary proof of concept that it is possible to apply reinforcement learning to this problem and to use the FaIR simulator to do so. I have demonstrated that RL algorithms are capable of learning reasonable emissions policies under this regime. However, there is a lot of room for improvement with this work. I would like to investigate experiments in multigas-mode that take into account emissions of other gases as well. I would also like to research reward shaping more and figure out how to properly design a reward function for this task that will allow the model to minimize the amount of emissions reductions necessary while still hitting temperature goals. The reward function design proved to be one of the most difficult parts of this project, and as this was my first foray into reinforcement learning, I am still learning how to design the reward well. I suspect that finding the right weighted-mixture of a temperature-based and cost-based reward might be a delicate process that could involve an entire research project in its own right.

References

Climate change: Global temperature. https://www.climate.gov/news-features/understanding-climate/climate-change-global-temperature.

The paris agreement. https://unfccc.int/process-and-meetings/the-paris-agreement/the-paris-agreement.

Co2 emissions. https://ourworldindata.org/co2-emissions.

A degree of concern: Why global temperatures matter. https://climate.nasa.gov/news/2865/a-degree-of-concern-why-global-temperatures-matter/.

Global greenhouse gas emissions data. https://www.epa.gov/ghgemissions/global-greenhouse-gas-emissions-data.

Peter cox tweet. https://twitter.com/coxypm/status/1469268373649473541/photo/1.

Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643*, 2020.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017. URL http://arxiv.org/abs/1707.06347.

David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *International conference on machine learning*, pages 387–395. PMLR, 2014.

Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. *CoRR*, abs/1602.01783, 2016. URL http://arxiv.org/abs/1602.01783.

R. J. Millar, Z. R. Nicholls, P. Friedlingstein, and M. R. Allen. A modified impulse-response representation of the global near-surface air temperature and atmospheric concentration response to carbon dioxide emissions. *Atmospheric Chemistry and Physics*, 17(11):7213–7228, 2017. doi: 10.5194/acp-17-7213-2017. URL https://acp.copernicus.org/articles/17/7213/2017/.

Christopher J Smith, Piers M Forster, Myles Allen, Nicholas Leach, Richard J Millar, Giovanni A Passerello, and Leighton A Regayre. Fair v1. 3: a simple emissions-based impulse response and carbon cycle model. *Geoscientific Model Development*, 11(6): 2273–2297, 2018.

Fair. https://fair.readthedocs.io/en/latest/index.html.

Detlef P Van Vuuren, Jae Edmonds, Mikiko Kainuma, Keywan Riahi, Allison Thomson, Kathy Hibbard, George C Hurtt, Tom Kram, Volker Krey, Jean-Francois Lamarque, et al. The representative concentration pathways: an overview. *Climatic change*, 109(1): 5–31, 2011.