Guided Reinforcement Learning

Visual Analytics Final Project

Grant Fennessy

Project Title: Guided Reinforcement Learning
Code: *https://github.com/gfenn/vaml_guided_learning*

Description

This project seeks to create a web-based tool for visualizing the progress of a reinforcement learning model during a long running training session, typically on the order of several hours to several days. The tool will present at-a-glance model training progress, attained reward, and estimated behavior quality. Users can directly interact with the model hyperparameters, and mark interaction events clearly so as to identify if their interactions are having positive or negative effects on rewards and behaviors.

Background

Training large neural networks can be incredibly slow, demanding a high degree of confidence before kicking off on a day, week, or even month long training session that has no guarantee of producing the expected results. I faced this exact problem with my thesis, which assembles a simple autonomous driving architecture capable of navigating a vehicle around a small neighborhood within a simulator. A full training run can take 2-5 days depending on hyperparameters, and this is considered short in comparison to many other papers on the topic, which can train for weeks or months. Further, since train duration is a hyperparameter, it can be difficult to know if more training is necessary (or worthwhile).

Reinforcement learning (RL) uses a reward function in an attempt to let a model learn on its own as it explores the world. As a result, even a perfectly optimized policy has no guarantee of producing the desired behaviors. This yields a model evaluation which requires manual observation beyond just the total reward attained - the analyst must observe the behaviors to see if they are as expected, and is responsible for adjusting the reward function as necessary to reach the desired behaviors. After all, models are trained to produce specific behaviors, not to attain an artificial notion of rewards. This conundrum makes evaluation sets more subjective and thus difficult to assess.

Most RL agents act within a simulation, and many simulations are nondeterministic. This means that all model comparisons are really apples to oranges, as two models can take completely divergent paths and thus must be evaluated subjectively: was the reward function higher? Were the behaviors better? These problems lead to validation set runs during training to be only of limited utility. While they may provide a bit of insight into the model's evolution, they don't always, without manual observation, paint a clear picture as to emergent behaviors. Models that take a long time to train often take a long time to validate as well, so a delicate balance is in play for validation: increasing validation set size increases training time, but lower validation sizes might not provide much or any insight.

Few tools exist for properly visualizing the state and quality of RL models. Though some techniques exist for more generic models, such as DeepEyes[1] which views the activation functions of individual neurons, there is a general lack of creative ways in which the RL reward function to desired behaviors can be visualized.

The Model and Data

The underlying model in this approach is an end-to-end RL agent that controls a vehicle within a simulator (Carla[2]). The model is a dueling deep-Q network[3] with a custom convolutional neural network based on VGG-13[4], and each step produces an output specifying if the vehicle should accelerate, decelerate, or turn[5]. A reward function was constructed in an attempt to train the agent to drive around the simulated neighborhood at as close to 25km/hr as possible, all while staying in lane and avoiding collisions.

In the first phase, a newly initialized model is trained with a learning rate of $1e^{-4}$ for 1 million steps, training every 6 steps. A second phase is then launched, training the model with a learning rate of $1e^{-5}$ for 2 million steps, training every 18 steps.

As the model runs, the reward obtained every step is collected and stored. Due to the large number of steps taken each training run (3 million), rewards data from each step are aggregated into buckets of 1,000 steps, thus reducing the "effective" steps to 3,000. Most of the data processing is just aggregating the data into these buckets, then running exponentially weighted moving averages to smooth out the results for display.

Predictions are run against all saved samples every 10,000 steps, and they are not aggregated or smoothed in any way.

Reward data and predictions are stored adjacent to the web application so that they can be queried on demand. Since the model is unfortunately not connected directly, running a new episode will simulate the model training by feeding data from a "feeder" folder into the valid data folder.

---

[1] N Pezzotti et. al. DeepEyes: Progressive Visual Analytics for Designing Deep Neural Networks. In *IEEE Transactions on Visualization and Computer Graphics*, pp 98-108. Vol 24, No 1. January 2018.

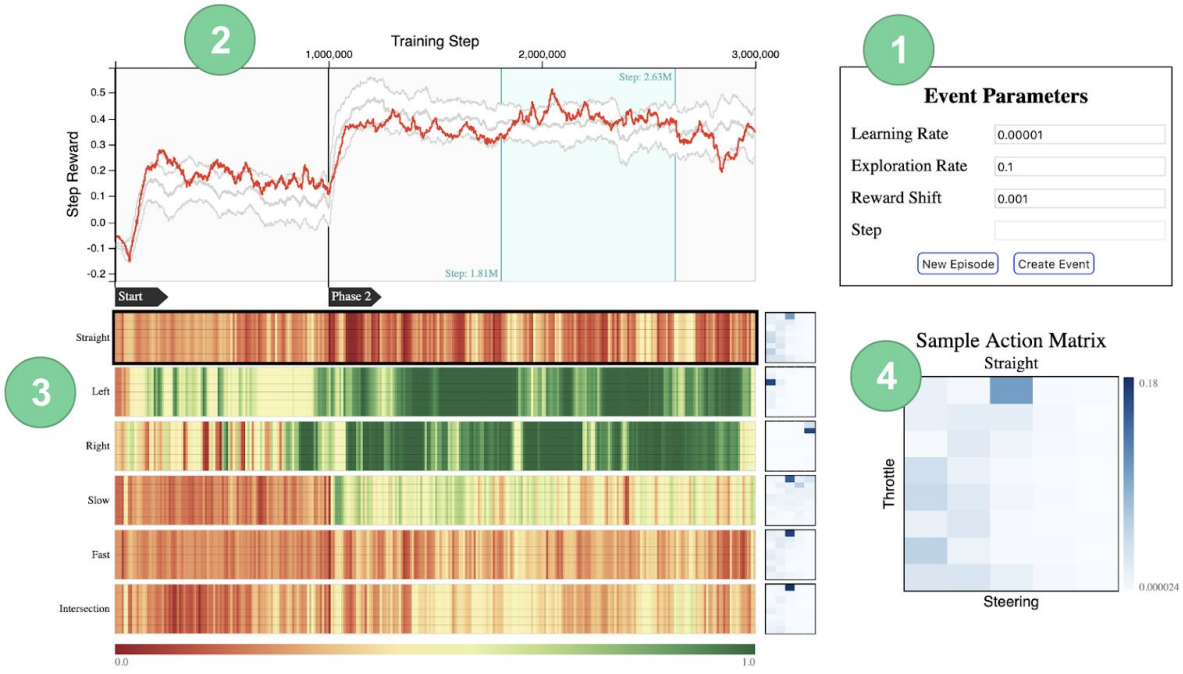
[2] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Kotlun. CARLA: An Open Urban Driving Simulator. In *Proceedings of the 1st Annual Conference on Robot Learning*, pages 1-16, 2017.

[3] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. Nature, 518(7540):529– 533, February 2015.

[4] Karen Simonyan and Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. arXiv:1409.1556 [cs], September 2014. arXiv: 1409.1556.

[5] Fennessy. Autonomous Vehicle End-to-End Reinforcement Learning Model and the Effects of Image Segmentation on Model Quality. https://etd.library.vanderbilt.edu//available/etd-03122019-204330/

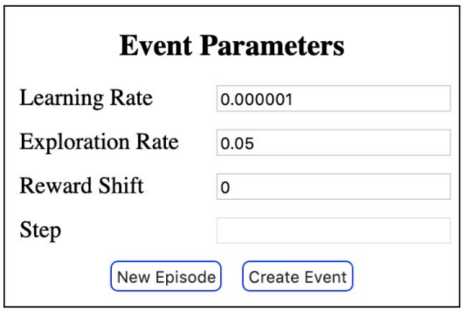
Approach



The application is split into four sections:

1. **Event Parameters**. Allows changing of model hyperparameters during training, along with kicking off new training runs.
2. **Rewards Graph**. Presents the current model's pre-step rewards, along with a representation of previous run results.
3. **Prediction Spectrograms**. Presents a list of samples, along with predictions made by the model against those samples during each training step.
4. **Action Matrix**. Displays the actions taken by the model at the selected step and selected sample, allowing analysis into what decisions the model is making.

Events Metadata



The creation of events allows direct interaction with the model, thus completing the concept of guided learning. Unfortunately the model is not connected to the application in this final build, and this metadata is mocked. If the backend and the model were properly synchronized to allow launching of and interaction with the model during training, this section would be the portal to those changes.

Creating an event allows modifying hyperparameters of the model, or even arbitrary model

training-rig options. An example would be controlling where the agent starts in any given episode. If a model is failing to learn right turns, for example, an option could exist in this portal that allows users to always start the vehicle in front of a right turn, and end the episode after a few hundred steps (long enough to complete the turn). Once the agent has attained more reasonable results for this behavior, the starting parameters can be reset to default.

This direct intervention with the model would allow a sort of student-master relationship, where the model builder acts as the master. If the student is performing a behavior poorly, they can be asked to practice it intensely in hopes that they develop that skill more quickly.

Rewards Graph

The rewards graph has two channels: the training step (x-channel), and the step reward (y-channel). Acting as a time series, data points can be plotted on the graph to form a line. Each point represents the average of 1,000 step rewards, and is further smoothed out by an exponentially weighted moving average of 3% to limit the extreme jagged lines that are all too common with rapidly changing neural network results.

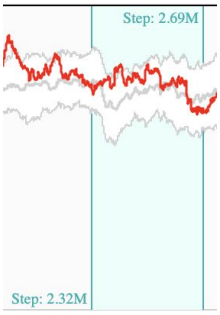
Eight ticks are placed on the y-channel to allow a sufficient understanding of the scale, and 3 ticks are provided for the x-channel to give a rough idea of the step (specific step is less important than specific reward). The graph itself has a light gray border and an even lighter gray fill so as to visually separate it from the otherwise white background of the application.

The main point of focus is a single red line, representing the current training run. The red provides sufficient contrast from the underlying colors to be very obvious to users, and also acts as a darker and thicker line so as to pop even if the visualization is in black and white.

Beneath the current run line is a curved boxplot[6] that displays the running 25th-75th percentile of previous training runs. A white channel represents this region, bounded by light gray lines for the 25th and 75th percentiles, along with a roughly central line which represents the 50th percentile. Encodings for this curved boxplot were selected to make it the lightest part

[6] M. Mirzargar and R. T. Whitaker. Curve Boxplot: Generalization of Boxplot for Ensembles of Curves. In *IEEE Transactions on Visualization and Computer Graphics*, pp. 2654-2663. Vol 20, No 12. December 2014.

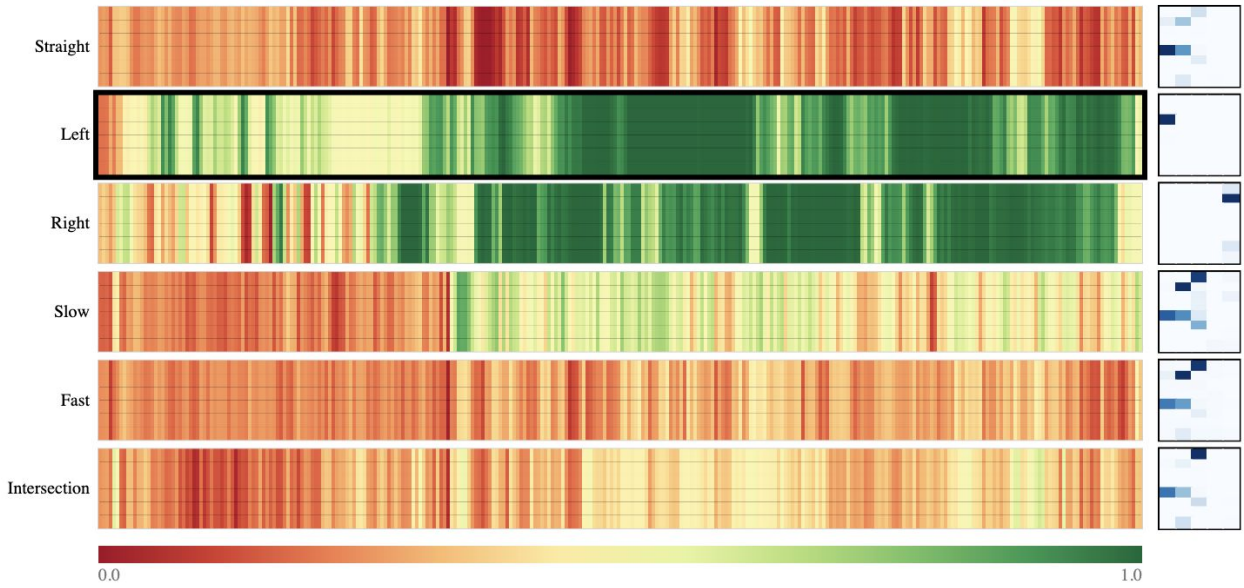
of the application so it feels like it is “carved out of” the baseline graph, while the main line is “overtop of” the baseline graph. This works out well, as the many colors do not compete for your attention even though a significant amount of information is presented on the graph.



Clicking anywhere on the graph will select that step on the graph, generating a light cyan vertical line. The selected step value is presented along the base of the graph adjacent to the line, using the same color to make the connection clear. Clicking and dragging (or shift clicking) will select a region of steps, bounded on each side by a cyan vertical line. The inside of the selection is a lighter cyan. Notice how the selection lines and region appear underneath the white curved boxplot channel, so as not to cover that information. It also makes the boxplot “pop” out even more. To make sure the selection information is always visible, an additional 10% of the maximum reward value is added to the top of the graph, and 10% is added to the bottom.

Events exist on the map in the form of a vertical black line at the event’s step, and a flag shaped glyph which contains the name of the event. Clicking on the event flag will select it, both selecting that step on the graph and selecting the event and updating the data presented in the Events Metadata screen. Shift clicking on an event allows region selection.

Prediction Spectrograms



The prediction spectrograms map out the “correctness” of each behavior: Straight, Left, Right, Slow, Fast, and Intersection. Correctness is a function of the model’s predictions for that step and the user annotated labels for that sample (0 points for a “bad” action, 0.5 for a “neutral” action, and 1 point for a “good” action). The points for each action (a probability density function summing to 1) are summed to yield the correctness score between 0 and 1.

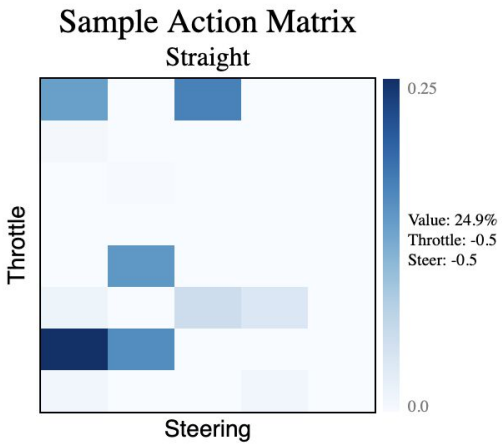
The main region has a categorical y-channel, where each item is one of the available samples. Along the bottom is the correctness scale. Within each row is a categorical

x-channel, where the first column contains the sample name, the second column contains the sample's correctness measurements for several runs, and the final column contains a small actions matrix for that sample (based on the currently selected steps from the rewards graph). Clicking anywhere on a row will select that sample, highlighting the spectrogram in a thick black box.

Spectrograms have a y-channel representing the run (in this case 6 training runs, but all rows currently leverage the same data so they are identical), and an x-channel representing the step, which is on the same scale as the rewards graph above it. All of the data points are represented as boxes which are colorized from dark red to light yellow to dark green based on the correctness of that run/step/sample set. Given that we naturally associate green with good and red with bad, these spectrograms allow for a very rapid understanding of sample correctness, along with identification of trends. The eye also rapidly detects outliers if a section of a certain color has a blip of a different color.

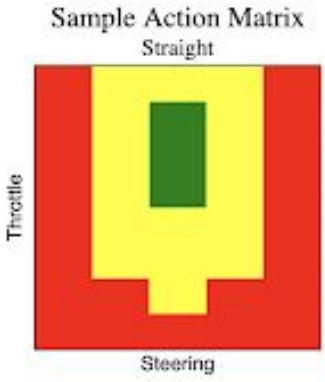
Actions Matrix

The predictions made for the selected sample at the selected step are presented in the action matrix view. The name of the selected sample is placed on top of the matrix for clarity. The y-channel maps the 8 possible throttle positions of the model (values from -1.0 to +1.0), and the x-channel maps the 5 possible steering wheel positions of the model (values from -0.5 to +0.5).



A given data point is marked by a box at the appropriate grid location, and is colored blue. The luminance is then adjusted based on the model's desire to take that action. Luminance is applied from brightest at the minimum value to darkest at the maximum value. The scale can be seen on the right of the graph.

If a range of steps are selected, the predictions for all of those steps in that range are normalized, allowing analysis of larger swaths of training. Hovering over an action with your mouse will display additional information on the right of the matrix: the value (probability of selecting that action), the throttle, and the steering position.

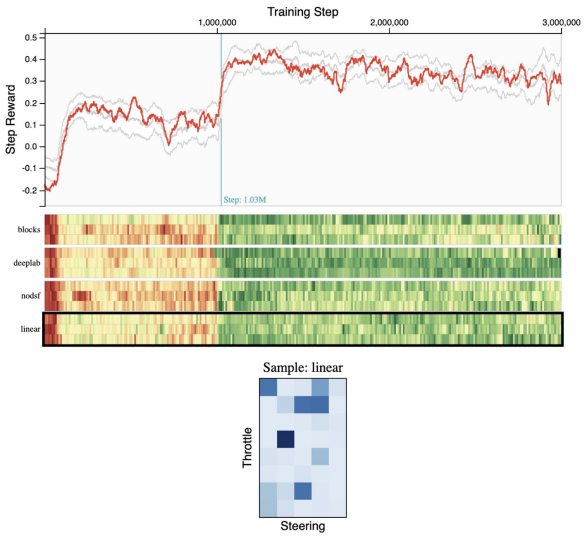


To see the correctness labels for each action, the user can click and hold anywhere on the action matrix. While holding down the mouse button, the matrix is updated to display the annotations for each action, where red cells are the bad actions, yellow cells are neutral actions, and green cells are good actions.

When hovering over an action, the steps during which that action was prioritized are highlighted on the rewards graph. Instead of a constant red line encoding, each point is encoded with a luminance from light to dark, where light represents no

desire to take that action, and dark means the selected action is the most desired action for that step. The color draws from the same color spectrum as the action matrix.

Comparing to the Baseline



The core components of the baseline remain in the final version of the application, but the parts are reorganized and generally improved.

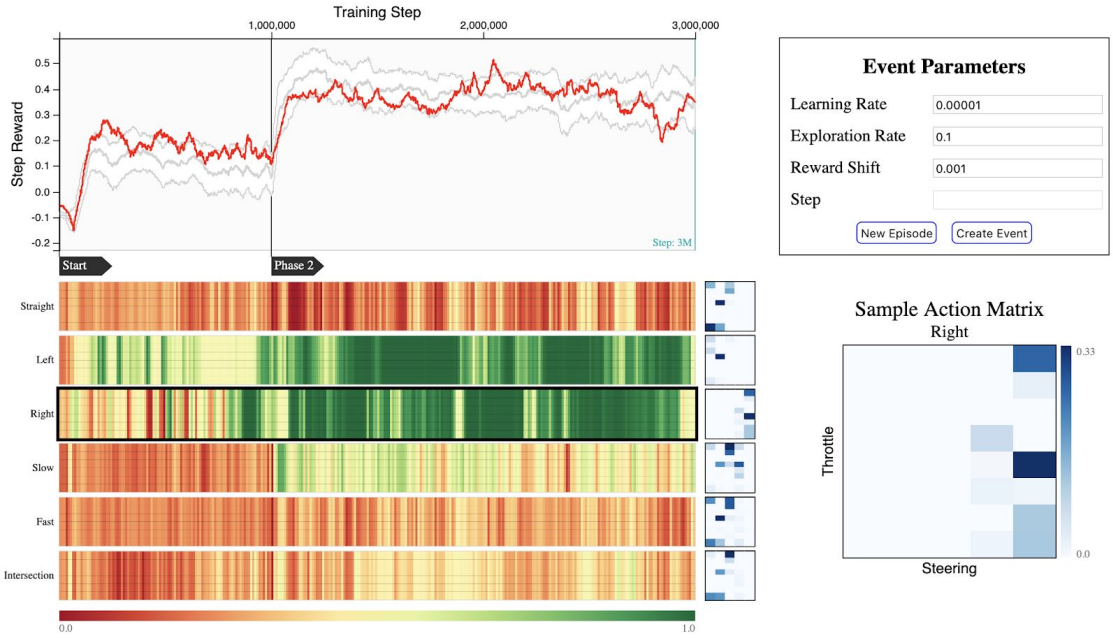
The rewards graph was upgraded from the baseline to include region selection, event markers, and being able to hover predictions to see where in the graph those predictions take place. These upgrades yield significant improvements in the analysts ability to explore the model's behavior during training.

The spectrograms didn't change directly, but the mini prediction boxes were added to the right of each row to allow easier digesting of predictions taken for each sample type. A scale was also added to more clearly understand the color values.

Finally, the actions matrix was converted to a square, a scale was added, along with the capacity to hover over an action so as to highlight the steps in the rewards graph.

In all, these improvements (along with the ability to make and view events) provide users with stronger tools for exploring the model's learning and behaviors.

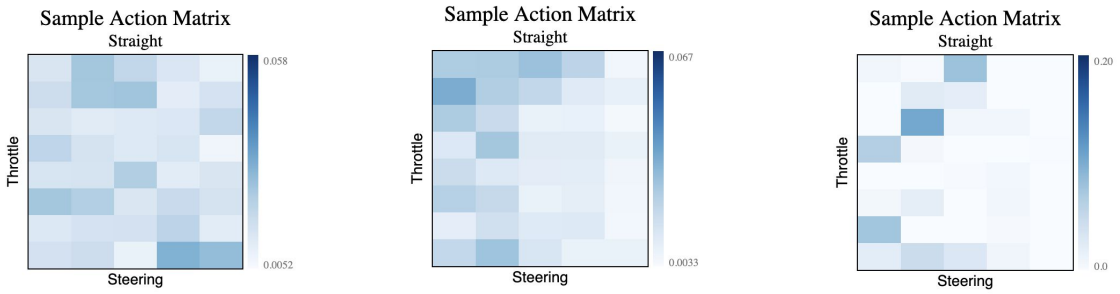
Results



Starting with the rewards graph, we can see that the model learned very quickly (about 200k steps), but learning seemed to stabilize or even slightly drop. At the beginning of the second phase, a similar pattern is exhibited. Just from the rewards graph, we could conclude that the length of phases 1 and 2 could be reduced, as they are not yielding significant improvements. We can also see that the curve boxplot shows that all of the models are yielding fairly similar results across training runs, so the training variance is low (which is a good sign).

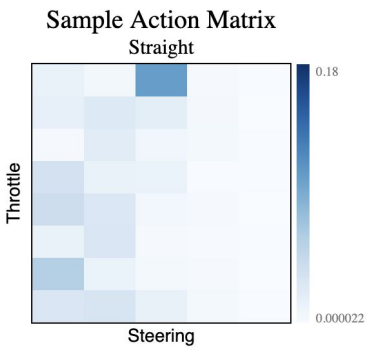
The behavior spectrograms tell a similar story, but they also make it clear that many of the behaviors are oscillating instead of stabilizing on a given (correct or incorrect) result. Interestingly, the model can't decide what to do during straightaways, unsure about either going forward at full speed or taking a braking left turn. Given the issues with "fast" and "slow" samples, this could lead us to the conclusion that the model isn't learning speed well.

Looking at the actions matrix, another story develops:

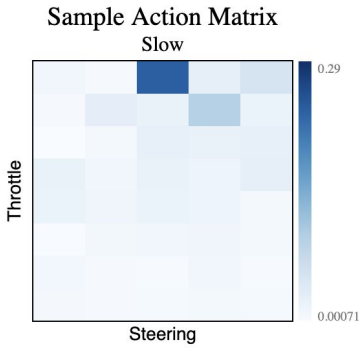


Steps 0k-100k | Steps 500k-600k | Steps 2m-2.1m

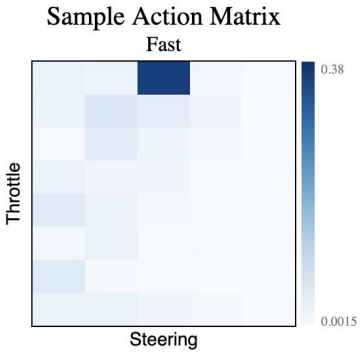
Even though the “correctness” of the Straight sample wasn’t steadily improving during training, the predictions over several 100k-step snapshots of the action matrix show that the model is slowly getting more confident, but favors left turns. The fact that the model is prioritizing a full throttle drive forward is also interesting, as the vehicle at this speed should be using a slightly lower throttle.



Straight



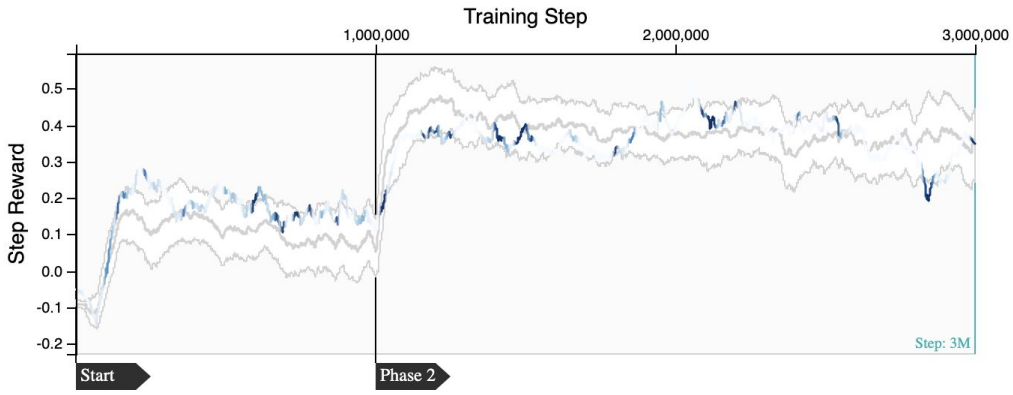
Slow



Fast

Comparing the Straight behavior to the Slow and Fast behaviors (in range 2m-2.75m), we can see that all of them have a propensity to accelerate the vehicle at full speed. It’s possible that the model is being trained with too high of a learning rate early on, thus slightly overfitting on high throttles. A potential solution to this problem is to augment the e-greedy exploration rate such that, early on in training, exploration steps have a high likelihood of taking the action predicted by a previously trained model instead of just picking random steps. This will act almost as a hybrid between reinforcement learning and imitation learning, where a “young” RL model is imitating an “old” RL model. Getting the “young” RL model to learn proper behaviors earlier on could in theory help prevent overfitting to poor behaviors. This idea stemmed directly from seeing these results visualized.

When mousing over the full brake and full left run action on the Straight sample, we see some fairly surprising results: selection of that action seems to be randomly distributed across many parts of the training process.



We would expect the model to sort of “rule out” that action as an unreasonable action, yet it seems to keep coming back as a plausible option. This suggests that the model is oscillating on action choices during straightaways, and can’t seem to figure out what actions are beneficial and which are not. This likely means that the reward function or the future reward gamma needs to be tightened up to make it more clear to the model which actions will produce higher strong rewards. Given that the vehicle drove for 3 million steps, it should at least know that slamming the brakes while pulling hard left isn’t going to do us any good! Without the ability to visualize when and how the model is taking certain actions, this analysis would be much more difficult.

Limitations

The action matrix demands 2D actions to be easily digestible. If the 40 output actions were entirely unrelated, the matrix would be much less clear with a glance, as the user would have to highlight the actions to recall what exactly they are doing. Additionally this can only scale to so many output actions in each dimension before the prediction boxes become too small to interact with.

Spectrograms show results for all runs, which becomes less valuable when there are a significant number of runs. An alternative approach would be to only show the current run, but allow many samples of a certain sample class to exist as rows within a spectrogram, allowing access to a wider behavior spectrum.

The events metadata region would need to be moved and expanded if a non-trivial number of hyperparameters was to be included. This section is also tricky since it will have to be custom tailored to the specific model being trained.

Conclusion

This tool would have been incredibly valuable for my thesis. Looking at the data now, there’s a long list of potential tweaks and improvements that could be made to the model, along with ideas for novel techniques in training the underlying RL model. If populated with a longer list of behaviors, it would become even more clear exactly how the model learns during training.

If additional data was available, such as a wider swath of “Straight” samples to draw from, the visualization would be even more helpful, allowing rapid analysis of which behaviors are generally improving and which are struggling. The addition of a tool that would make creating and annotating these samples simple for users would make this application very powerful indeed.

Despite the limitations, this approach, along with the high degree of interactivity provided, allows model builders to watch in real time whether or not the model is learning the underlying behaviors, and potentially take actions in real time to intervene. Insight into exactly how the model is failing to learn can help make each training iteration better than the last.

This visualization should prove invaluable to anyone working with reinforcement learning models, especially those that take a significant amount of time to run.