Guided Reinforcement Learning

Visual Analytics Project Proposal

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 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, allow exploration of progress across several parameters, provide a way to directly interact with the hyperparameters, and mark interaction events clearly so users can identify if their interactions are having positive or negative effects on model training.

Background:
Training large (convolutional) neural networks can be incredibly slow, demanding a high degree of confidence before kicking off on a day, week, or month long training session that has no guarantee of producing the expected results. I've been facing this exact problem with my thesis, which is attempting to assemble a simple autonomous driving architecture that is 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 yield 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, 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], viewing 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.

Novelty:

This tool will provide an observation suite that makes model training progress (or lack thereof) more clear to analysts, allowing for direct interaction with the training, early termination for ready models, or abandonment for models that aren't making sufficient progress in an appropriate amount of time. Ultimately the tool will result in less wasted time during model training, as often times experiments with hyperparameters or architecture configurations have a reasonable likelihood of producing worse off models, and identifying that as early as possible to minimize time wasted on these experiments will result in attaining a higher quality model in less time.

Interactions between changing hyperparameters mid-train, along with a way to visualize the impact of those deltas, is a novel technique that could allow for fewer train iterations and more quickly reaching a highly capable model. Instead of finding and locking in static "optimal" hyperparameters before model training begins, the hyperparameters can be reconfigured on the fly as progress is made. Further, the analysts could be able to create "scripts" with these interactions off of simple triggers or at specific train steps, allowing them to develop a hyperparameter guide that increases the likelihood of the model learning necessary elements in the right order, thus streamlining the entire training process.

With respect to reinforcement learning, the tool will provide a way of directly analyzing model behaviors with respect to user-applied changes in training conditions. Beyond just the reward function with respect to time graph, analysts will be able to observe pre-identified behaviors and if they are being developed properly.

Completed episodes by the model will be stored for later viewing, allowing analysts to observe the scene and actions taken. Clips of these episodes can be identified and stored as "behaviors", and the action space can coarsely be annotated by the analyst in terms of desirability of the possible actions: likely good (+1), neutral (+0), and likely bad (-1). For each transition, the softmax of the outputs will be multiplied by the annotations to produce a single action "quality" metric from -1 to 1.

As the model (or even future models) continue to train, predictions will be run against cached episode clips, and results will be presented in terms of how the model predictions in the given scenario line up with the annotated actions. Since the model isn't training against this information, the clips can be added/removed or (re)annotated at will without impacting model quality. Instead, they can be tweaked at will until the analyst feels as though the collection of clips represents the behaviors they're looking for. At that point, the quality metric becomes a very strong indicator of model success, completely detached from reward attainment.

---

[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.

Data:

While running the simulation, the following pieces of data will be collected: reward earned each step, total reward for each episode, and number of steps taken each episode. Realistically there will be millions of data points collected here, so some degree of aggregation will be necessary to compress the data into a reasonable footprint.

Episode observations and predictions for each step are stored locally, allowing them to be viewed later by the analyst. All of this data needs to be saved to disc for later use, and if episode data is too taxing on disc space, the number of stored raw episodes can be limited, writing over older files as space fills up. Saved episode clips and user defined annotations must also be stored to disc, but are not overwritten. Clips must store the observations and annotations for each action in the action space.

All user interactions with the model are stored as events that have timestamps and a list of actions taken.

Importantly, the tool will be web based which means data needs to be sent to a web browser. This imparts a limit on how much information can be sent and stored in memory for the visualization tool, and thus will require data to be minimized whenever possible, and re-queried on demand instead of held persistently.

Baselines:

Given the web-based nature for this project, a combination of D3, React, and TypeScript appears to be the best combination of tools to handle the drawing of visualization, as they provide extensive drawing capability, a fairly straightforward interaction schema, and the capacity to readily integrate user actions with model and visualization changes. If any real-time components are to be established, such as viewing data as it comes in (instead of requiring a manual refresh or requery), websockets would likely be the tool of choice for connecting the browser with the server.

The first component I plan to implement is a Curve Boxplot[2], which will likely prove useful when visualizing per-clip prediction qualities over time. Instead of showing many curves, the distribution of the curves can be displayed to make it more clear how the model is learning. Curve boxplots might also be valuable in other parts of the project, such as displaying reward with respect to time across multiple runs, migration of the actions taken for a specific clip over time, etc. As a result, mastering this widget early on will help provide more clarity to where and how they can be used to maximize effectiveness.

I also plan to implement a spectrogram component, which will in theory display model prediction quality for each clipped episode with respect to training time. Spectrograms represent a very different type of visualization from curved boxplots, which will help cement my understanding of D3, so a successful implementation of these components will allow me to implement the remainder of the project with high technical confidence.

---

[2] 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.

Schedule:

Feb 11th, Baseline Updates - Set up a demo web server that has a simple time plot widget using d3. The webpage will retrieve mock data from the server for presentation. I'd like to have one of the two components (curve boxplot or spectrogram) implemented or at least coming along well. The mock data should be matched up against what will eventually be collected from my actual RL model.

Feb 28th, Baseline Demo - The demo server is now connecting with a websocket, and is able to query for data updates at a regular interval. The backend servlet notifies the socket any time new data is discovered, and the updated data is presented in the visualization. Both the curve boxplot and the spectrogram widgets are completed and display the data retrieved from the server. Instead of using mock data, use data harvested directly from actual model training runs (no expectation yet that the server and the model training are hooked up).

March 20th, Project Update 1 - Configure my actual model architecture to communicate with (or be contained within) the server, allowing actual data to be presented in real time to the webpage. Record episode data so that the webpage can view previous episodes and save clips. Allow presentation of clips on the webpage. The spectrogram should be completely implemented, and a confusion matrix style visualization should display predictions made by the model at any given timestep in the episode viewer.

April 3rd, Project Update 2 - The webpage should support annotating the clipped samples, and the associated confusion matrices should be colorized based on prediction category. A visualization for presenting the model's prediction quality of a clip with respect to time should be designed and implemented using the annotation data. A field should be set up that allows changing a few simple hyperparameters. Any changes should be applied against the model, and event data should be stored and displayed on the visualization.

April 17th, Presentation - Final design aggregation completed, with all widgets in the correct places and functioning as expected. The two project updates schedules may by a bit overeager, so any components that couldn't be finished by the previous update will be handled during this phase. Evaluation of the project will take place by running a "naive" RL model, then attempting to run the same model while using the project to steer hyperparameters to see if any improvements can be made. Unfortunately since RL neural networks have fairly nondeterministic when it comes to training, it'll be fairly difficult to run a comprehensive evaluation of the project's utility given the available project window - many runs, each taking several days, would need to take place on the fully completed system to provide a strong indication as to how much of an improvement the tool actually makes.