Independent Study

Report-8(Final), 20th May

Submitted by: Vaibhav Garg(20171005), Kartik Gupta(20171018)

Recap

We tried to fit our framework on a vector-output based environment(Bipedal Walker) last time but weren't really able to get decent results. So we decided to take a step back and work on an environment with just one single continuous output since we had already successfully performed on a discrete output environment(Cartpole).

Results and analysis on MountainCar

We decided to go ahead with the MountainCarContinuous environment which requires a single continuous valued output.

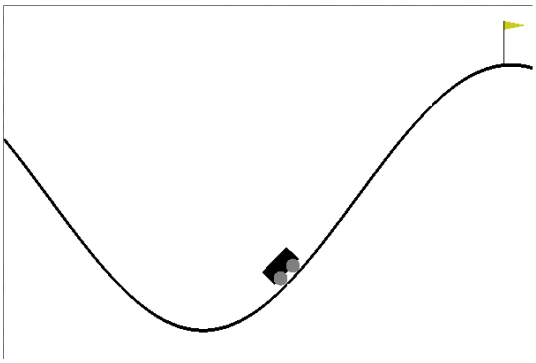

Fig: MountainCar Environment

For the oracle/teacher network we got a nice model online that used Q-learning algorithm with e-greedy policy and discretization and consistently gave a 90+ mean reward. We used an extremely simple sketch for the program generation as given below where <if> represents our program:

```
<if>;

<if> -> if <cond> then <act> else <act>;
<cond> -> <t0> + <t1> > <var0>;

<act> -> <t0> + <t1> + <var1>;
```

```
<t0> -> <var2>*peek(<input0>);
<t1> -> <var3>*peek(<input1>);

<var0> -> c0;
<var1> -> c1;
<var2> -> c2;
<var3> -> c3;

<input0> -> h0;
<input1> -> h1;
```

Our program generation module generated the following program using the above sketch:

```
if c2*peek(h0) + c3*peek(h1) > c0
then
    c2*peek(h0) + c3*peek(h1) + c1
else
    c2*peek(h0) + c3*peek(h1) + c1
```

Note: Same named parameters at different places represent completely separate learnable parameters, the program is generated in such a way since the grammar is written in that fashion, making way for easier recursion, etc.

Then we applied our parameter optimization module on this program to find optimal values for all the different parameters. We chose a simple/intuitive loss function to begin with:

$$Loss = (Oracle(obs) - prog(obs))^2$$

where obs is nothing but *[h0, h1]* for this environment.

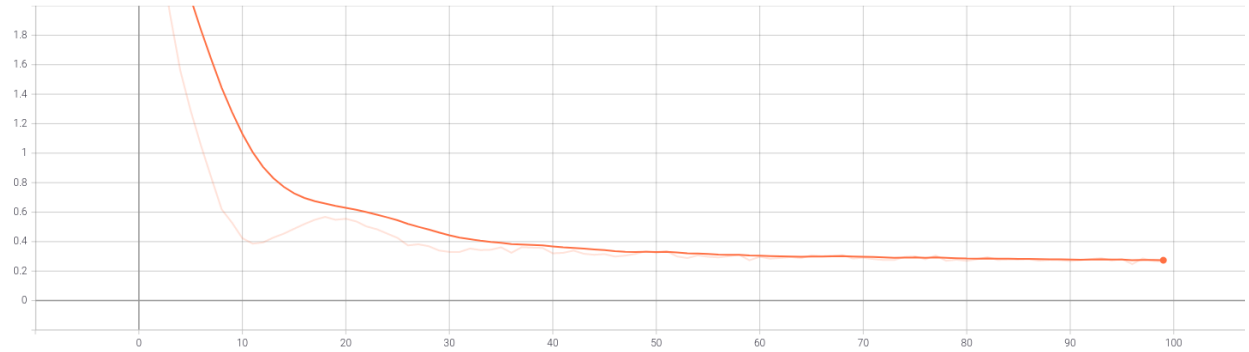

Fig: Decrease in loss with time

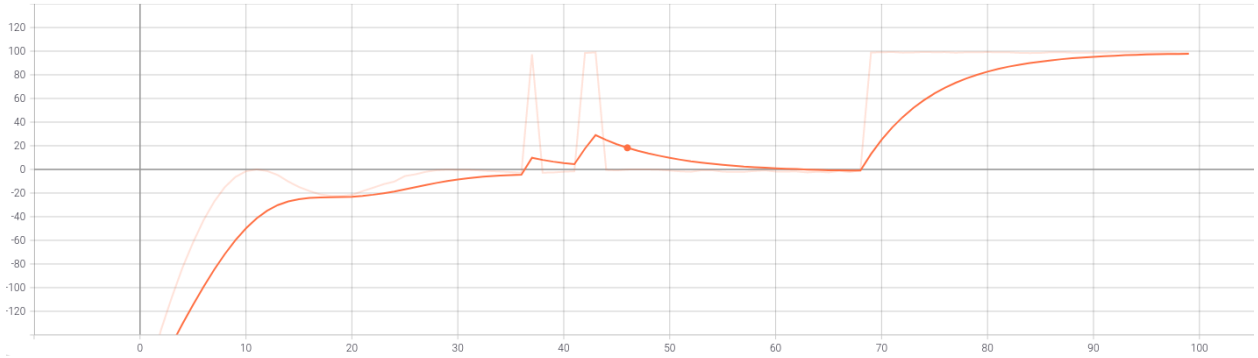

Fig. Increase in cumulative reward with time

We managed to achieve a decent training curve using just 100 episodes of training with a learning rate of 0.1 along with the TensorFlow Adam Optimizer. The final optimized program we got is given below which performs consistently with a **98+ cumulative reward**. One can also check its performance by simply using the program below(minor tweaks to convert into the Python syntax might be required). It is interesting to note that this program performs even better than the oracle it learnt from.

```
if 0.6*peek(h0) + 0.1*peek(h1) > 1.83
then
    0.39*peek(h0) + 6.69*peek(h1) - 0.08
else
    -0.19*peek(h0) + 7.8*peek(h1) - 0.12
```

Note: The peek() operator simply gives the latest observation from the particular sensor.

Clearly, our framework was able to do an excellent job in the MountainCarContinuous environment and hence given a decent enough sketch and oracle should similarly be able to perform on any continuous-output environment.

Bipedal Walker

Agent	Best Score
Oracle	241
Program	-50
Oracle with rounding	-49

Looking at the amazing performance on the above environment we tried many different variations of our experiment to get similarly good results on the BipedalWalker environment. Our

agent was able to take only a few steps before falling in the best case (-50 cumulative reward). This is significantly less than what the oracle is able to achieve (240+ cumulative reward).

The main suspect for this poor performance is the sensitivity of the environment to the inputs. We think that even very minor deviations in the inputs to the environment (ie, even minor deviations from the ideal actions predicted by the oracle) can cause the walker to fall over very quickly.

To test this hypothesis, we tried rounding the actions predicted by the oracle to the nearest tenths place (ie, an error of +-0.05) before sending them to the environment. This caused the performance to drop significantly (-49 cumulative reward) even for the oracle. We believe that the environment is too sensitive to afford an approximation with a program, since even small approximations have a significant impact on the performance.

Concluding Remarks

Through the course of this project we have provided a framework that can generate random programs from a given sketch, make a differentiable computation graph for it, and optimise the graph to learn good, interpretable approximations for deep RL models. This framework can be extended to any model even outside the realm of RL.

Further work can be done in investigating methods to make the model learn better in sensitive environments as well. More programming constructs can be incorporated so that we can make environment modeling (in the form of writing sketches)easier for domain experts.