Independent Study

Report-6, 18th April

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

Parameter Optimization

We use the input sketch grammar to generate a program parse tree. The sampled program has many constructs, as mentioned in the previous reports, but two of them are relevant for discussion here - Inputs and Variables.

Inputs are values that will be input to the program. These values are generally observation hystories, but we provide an implementation which is modular enough to allow for marking any terminals as inputs.

Variables are not the traditional programming variables (we don't provide support for those in general). Variables can be thought of as hyper-parameters, like thresholds or weighing factors, which we don't know at the time of specifying the sketch.

Once we sample a random program, with some Inputs and Variables among other constructs, the aim is to find the most optimal values for these variables, so that the generated program mimics the oracle as closely as possible. For this we take the parse tree, and for each node in the parse tree, we specify functions to convert the node into an equivalent tensorflow code. The Variables for the trainable parameters in the generated model. This allows us to use standard backpropagation to obtain the values of the most optimal program.

We would like to point out that we have slightly differed from the reference paper here. For obtaining the most optimal program, the reference paper uses bayesian optimisation. We opted for a tensorflow based machine learning approach because it can be extended to arbitrarily large programs and complex constructs.

Performance & Analysis on Cartpole

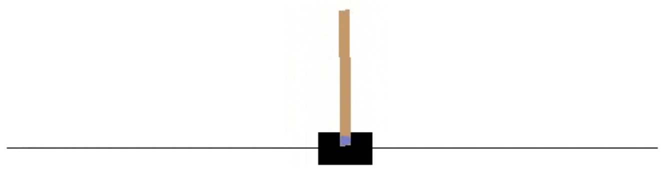

Fig: Cartpole Environment

As we had decided earlier, we ran our code on the Cartpole environment first. For the oracle/teacher network we had trained using a simple Monte Carlo algorithm that performed fairly well with a consistent cumulative reward of over 200. We used a quite simple sketch for the program generation as given below where <E> represents our program:

```
<E>;
<E> -> <if>;
<if> -> if <B> then <left> else <right>;
<B> -> <cond>;
<cond> -> <t0> + <t1> + <t2> + <t3> > <var5>;
<t0> -> <var1>*peek(<input1>);
<t1> -> <var2>*peek(<input2>);
<t2> -> <var3>*peek(<input3>);
<t3> -> <var4>*peek(<input4>);
<var1> -> c1;
<var2> -> c2;
<var3> -> c3;
<var4> -> c4;
<var5> -> c5;
<left> -> 0;
<right> -> 1;
<input1> -> h1;
<input2> -> h2;
<input3> -> h3;
<input4> -> h4;
```

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

```
if c1*peek(h1) + c2*peek(h2) + c3*peek(h3) + c4*peek(h4) > c5
then
    0
else
    1
```

Then we applied our parameter optimization module on this program to find optimal values for c1, c2, c3, c4 and c5. We chose a simple/intuitive loss function to begin with:

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

where obs is nothing but *[h1, h2, h3, h4]*

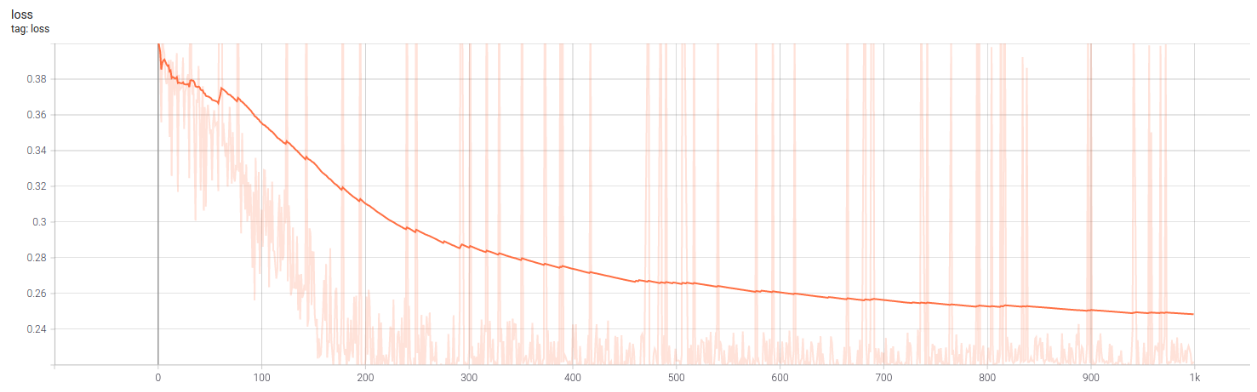

Fig: Decrease in loss with time

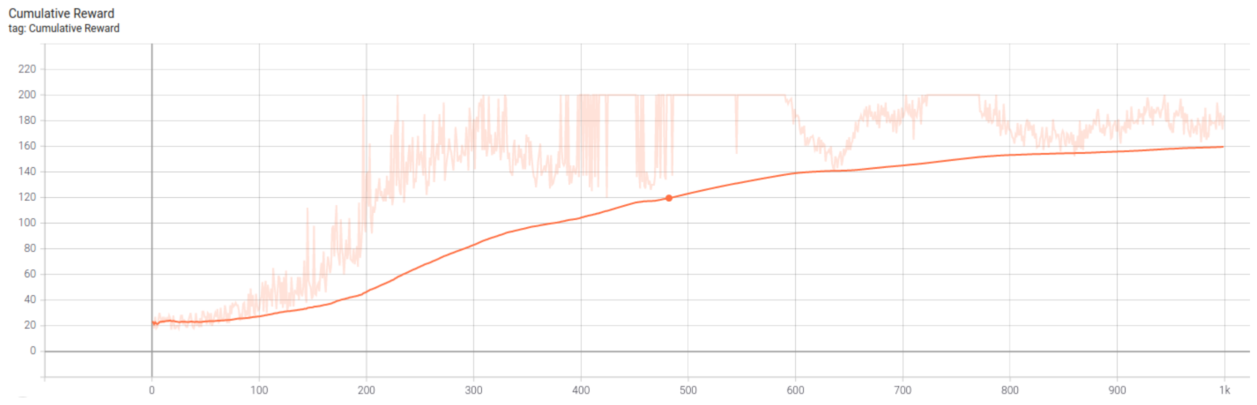

Fig. Increase in cumulative reward with time

We managed to achieve a decent training curve using 1000 episodes of training with a learning rate of 5e-3 along with the TensorFlow Adam Optimizer. The final optimized program we got is given below which performs consistently with a **180+ 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).

```
if 0.116*peek(h1)-0.686*peek(h2)-1.158*peek(h3)-1.434*peek(h4) > -0.345
then
    0
else
    1
```

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

One of the basic inferences we can make by just looking at the program is that the coefficient of h1(cartpole position) is very small in magnitude probably because it doesn't really affect the action much. Please note here that this point is valid even though there is no input normalization being done because the range of values of h1 is much lower than that of h2, h3 and h4.

For the Final Report

Since Cartpole was a simple environment to work on, we made no use of the the Neighbourhood-Pool module in our above experiment and were still able to get quite decent results. However, this probably won’t be the case for a Breakout-like environment. For the next and the final report, we’d like to evaluate our pipeline on the Breakout environment and present our analysis on that.