Imitation Learning in Tensorflow

09-25 14:09

Overview

Imitation learning, a.k.a behavioral cloning, is learning from demonstration. In other words, in imitation learning, **a machine** learns how to behave by looking at what a teacher (or expert) does and then mimics that behavior. An example can be when we collect driving data from human and then use that data for a self driving car.

Formulation

Imitation learning is a supervised learning where we have a set of observation and action pairs

\[\{(o_i, a_i)\}_{i=1}^N\] where o_i is observation that we collect from environment (sensory outputs), and a_i is the best (I use best very loosly here) action that machine can take (comes from expert). Similar to any other supervised learning, the goal in imitation learning is to estimate a function $f(\cdot)$ so that given an observation o, we estimate the *best* (defined below) action:

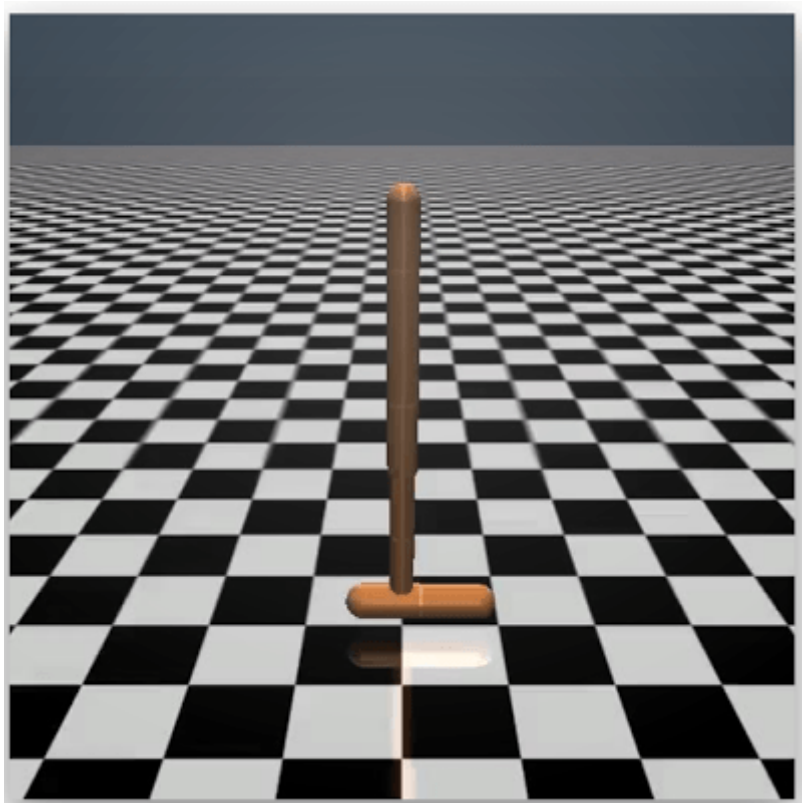
\[\hat{a} = f(o) \]

The lost function (to define *best* in above sentence) is defined to minimize distance between expert action and our estimation:

\[min \sum_i dist(\hat{a}, a) \] For this exercise I use Euclidian distance for $dist$ function above.

Gets our hand dirty

Let's use Hopper from openAI gym. The Hopper is two-dimensional one-legged robot. Its goal is to jump as far away as possible without falling. You can find more information about Hopper onOpenAI page



Create Dataset

Let's import gym package and setup the Hopper environment

```
import gym
env = gym.make('Hopper-v1')
```

In gym you can get the environment action/observation space by doing:

```
print(env.observation_space)
print(env.action_space)
```

For imitation learning, we need an expert to show us what action to take for each observation. For Hopper lets grab expert policy from here

Let assume we have load this pickle file in a way that

```
expert_action = expert_policy(obs)
```

To collect data from expert, we run the Hopper multiple times and each time we create pairs of (observation, action). Note that the environment is stochastic environement so every run is a bit different from others.

```
num_iter = 80
for i in range(num_iter):
    steps = 0
    while not done:
        expert_action = expert_policy(obs)
        observations.append(obs)
        actions.append(expert_action)
        obs, r, done, _ = env.step(action)
        steps += 1
        if steps >= env.spec.timestep_limit:
            break
```

In above code, in each iteration, Hopper jumps until it is done or it reaches certain step limitation.

Neural Network

Now that we have our supervised dataset, we can create a neural network to get an observation and estimate the action to take:

```
model = model = BCModel(log_name='./logs/'+args.envname,
                        observation_dim=env.observation_space.shape[0],
                        action_dim=env.action_space.shape[0])
model.train(expert_data)
```

The detail of the BCModel can be found on my github account . Here, let's just look at the computation graph:

```
def build_graph(self, activator=tf.nn.tanh, regularizer_scale=0.01):
        self.observations = tf.placeholder(shape=(None, self.observation_dim), dtype=tf.float32, name='observations')
        self.actions = tf.placeholder(shape=(None, self.action_dim), dtype=tf.float32, name='actions')

        # regularizer
        regularizer = tf.contrib.layers.l2_regularizer(scale=regularizer_scale)
```

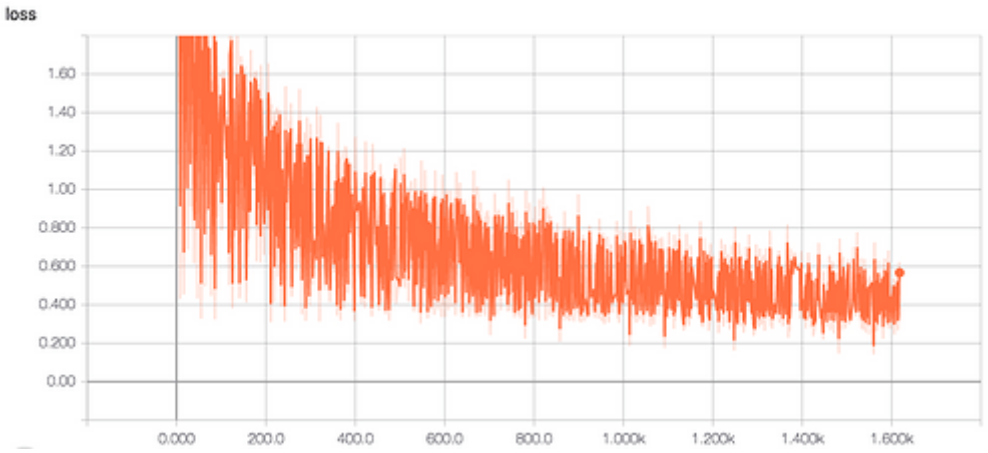
```
# layers
W1 = tf.get_variable(shape=(self.observation_dim, 128),
                     regularizer=regularizer,
                     initializer=tf.contrib.layers.xavier_initializer(),
                     name='W1')
b1 = tf.get_variable(shape=(1, 128),
                initializer=tf.contrib.layers.xavier_initializer(),
                name='b1')
logit1 = tf.matmul(self.observations, W1) + b1
layer1 = activator(logit1, 'layer1')

W2 = tf.get_variable(shape=(128, 64),
                     regularizer=regularizer,
                     initializer=tf.contrib.layers.xavier_initializer(),
                     name='W2')
b2 = tf.get_variable(shape=(1, 64),
                initializer=tf.contrib.layers.xavier_initializer(),
                name='b2')
logit2 = tf.matmul(layer1, W2) + b2
layer2 = activator(logit2, 'layer2')

output = tf.matmul(layer2, W3) + b3
output_action = tf.identity(output, 'output_action')

self.l2_loss = tf.losses.mean_squared_error(labels=self.actions, predictions=output)

grad_wrt_input = tf.gradients(self.l2_loss, self.observations)
grad_wrt_activations = tf.gradients(self.l2_loss, W1)

self.optimizer = tf.train.AdamOptimizer(learning_rate=self.LEARNING_RATE).minimize(self.l2_loss)
```

This is fully connected network with linear output layer. Hidden layers have similar activation function (activator).

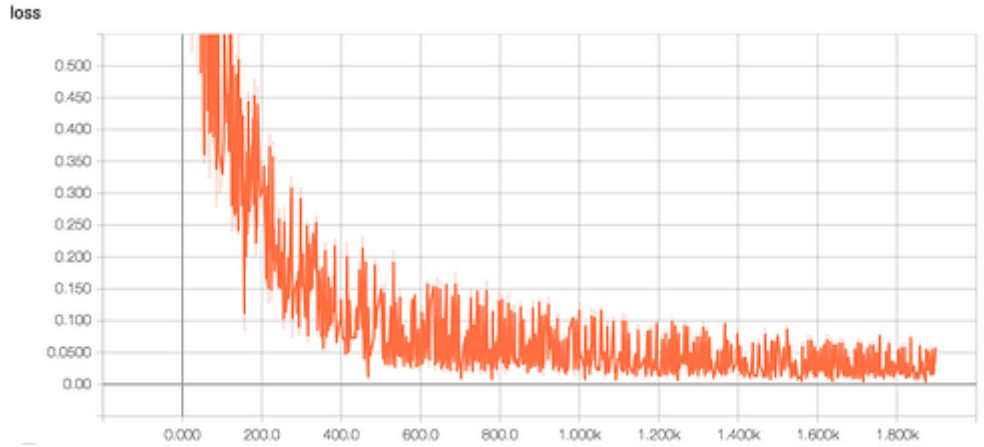
Findings

There are couple of interesting findigns while I was trying to get maximum reward for the hooper.

1- Adam optimizer works much better than gradient descent. The following plots shows the MSE error rate for identitcal networks with SGD and Adam optimizer. As one can see 1) Adam optimizer reaches to lower error faster 2) it is more stable (less variance) compare to SGD



l_2 loss minimization using SGD



l_2 loss minimization using Adam optimizer

2- tanh works better that relu for activation function. In the following table I summerize different attempt by changing activation, optimizer, and output layer. Superisingly, LSTM output layer was not giving better reward than linear layer. One possible explanation is that it increases the model complexity and amount of data might not be enough for network to learn. This can be some of future works.

原文链接：http://hameddaily.blogspot.com/2017/09/imitation-learning-in-tensorflow-hopper.html?utm_source=tuicool&utm_medium=referral

标签： TensorFlow