6.867 Machine Learning Project Ideas

November 12, 2015

1 Reinforcement Learning for Quadrotor Flips

Use reinforcement learning to teach a quadrotor how to do a flip.

1. First teach it how to do a flip.
2. Next see if it can do obstacle avoidance using output feedback with the quad???

2 Q-Learning for Autonomous Driving

Basically want a car that can drive autonomously through a simple obstacle field. Alternatively could use Policy gradient (PG) methods.

2.1 Motivation & Big Picture

- **Demo:** Car equiped with LIDAR sensor autonomously driving through obstacle field without crashing.
  - Project would be in simulation.
  - If that works would like to do it on the actual car that the UROPS are building.
- Think about simple case first, then add all the bells and whistles.

2.2 Technical Approach

Want to apply Q-Learning techniques. To do this need to formulate our problem as a dynamic programming problem.

- **State Space:**
  - For simple case, no states needed to describe the motion of the car, i.e. driving at constant speed.
  - Let $X = (x_{-n}, \ldots, x_0, \ldots, x_n)$ where $x_j \in \mathbb{R}_+$ is the range returned by the nth laser beam.

- **Action Space:**
  - Simple Model: The action set is just $\mathcal{A} = \{L, R, S\}$ just L=Left, R=Right, S=Straight options.
  - Complex Model: Allow for the car to have some state. Use a Dubins car model, so velocity and steering angle θ are the state variables corresponding to the car.

Now we have specified the state and action space. To apply standard Q-Learning would need to discretize the state space so we could think about $Q(X_i, a)$ for each state i.

- **Representations of Q-values:**
  - Simple Case: Use few beams with discretized range measurements so that we can actually discretize the state space.
  - In order to scale up we (probably) need to do something smarter than representing Q values with a grid. Suggestions offered in Gosavi survey paper. My intuition is that we should be able to do well enough with the linear representation outline below. Don't need to go to something as complicated as a neural network.
    * **Linear:** $Q(X, a) = \sum_j w_a(j)\phi_j(X)$ where ϕ is a feature map. This is a linear representation of the Q-values. Makes intuitive sense to me. Have an idea of what the ϕ values could be. ϕ_j only depends on beam x_j and is a trucated version of $1/x_j$. Short laser detection ranges are bad, it means we are very close to an obstacle.
      · If we allow speed to be in the state space, how does affect this representation???
      · Would speed enter into the $\phi(X)$?
    * **Neural-Network:** Could train a neural network to represent Q-values.
      · I think this is going to be much harder to train.
      · Could get stuck in local minima, hard to know if we have a bug or it's just not converging. Going to try to stay away from this for the moment.

2.3 Topics & Questions to Discuss

- Compare/contrast performance of the different strategies outlined above.
- Effect of having a better sensor, i.e. more beams. How does it affect . . .
  - Training time - presumably makes it longer
  - Performance - is it any better? much better?

- On policy vs. off-policy learning.
- Even if the true model of the car is quite complex, i.e. Dubins, is training with simple actions L, R, S and then mapping those to the full car model enough?
  - As a follow up, what is the right "simple space" to train the Q-learning on, that abstracts from the full car dynamics, but still captures the essentials, enough so we don't crash into stuff?
- How does the type/structure of environment that we train on affect performance?
  - Train with convex obstacles, then what will be performance with concave obstacles?
  - Train with concave, performance on convex???

3 Project Proposal

3.1 Steps/Milestones

1. 11/16 - Get simulation environment up and running
   (a) Raycast within director
   (b) Simple car model (in python) -
      i. At first car with constant forward speed and 3 actions, {Left, Right, Straight}
   (c) Auto-generation of worlds with random collections of obstacles and boundaries. Should have a few canonical worlds that we can always access for testing repeatability
   (d) Simple controller
      i. Maybe performs random actions for now.
   (e) Have a method for logging the data of the runs? Print to a file, or broadcast to lcm. Probably print to a file is better
2. 11/23 Q-learning (simplified)
   (a) Discretize the state space, e.g. 5 lasers with only 5 distances each, then size of state space is approximately 10,000. Not that large . . .
   (b) Drive around in very simple world, with no obstacles, except around the edges to keep the car from driving off to infinity.
   (c) Verify performance of the code and convergence of the Q-values
3. 11/30 Q-Learning (full)

(a) Represent the Q-values using linear regression, as outline above in section 2.2.

(b) Verify convergence of the Q-values in simple world.

(c) Add obstacles and verify convergence

4. 12/5 Evaluation of algorithm performance

   (a) How does changing the number of lasers and their field of view affect

      i. Training time

      ii. Performance

3.1.1 Time permitting goals/milestones

1. Generalize our approach to a more complicated car model

   (a) e.g. allow the car to have "state" instead of the state just being the laser state

2. Increase the number of lasers

3.2 Risks

1. Q-values fail to converge, or they converge very slowly

   (a) Maybe because we aren't sampling the entire space

   (b) The state space of our Q-values is too large . . .

   (c) **Mitigation approach:** Reduce the dimensionality of the space by applying some hand designed features to the laser data before it is processed by the algorithm.

3.3 Division of Responsibility

Pete:

- Simulator

Lucas:

- Q-Learning (simple)

Both:

- Q-learning (full)
- Algorithm evaluation
- Hand-tuned features for dimensionality reduction . . .