UROP project: Integrating Planning and Deep Learning

Peter Karkus
Learning and Intelligent Systems, CSAIL-MIT
karkus@mit.edu

Project

We aim to integrate model-free deep learning and model-based algorithmic reasoning for robust robot decision-making under uncertainty. The key to our approach is a unified neural network representation of the robot policy. This policy representation encodes both a learned system model and an algorithm that solves the model, in a single, differentiable neural network. For details please refer to our previous work on QMDP-nets and Particle Filter nets.

In this project you would investigate important research questions related to the combined learning-planning approach: 1) how do we transfer learned models from a simulator to a real robot? 2) which algorithms can be encoded in a neural network? 3) what robotic tasks can we solve by integrating learning and planning?

Through the course of the project you would explore one or two of these questions. You would be doing experimental work using state-of-the-art deep learning techniques, and potentially a real robot system.

Problem sets

This series of problems allows us to see if we are a good research match. First, they will give you a taste of the type of work that you would do in this UROP. Second, they will allow us to to see if you need more Machine Learning/programming/math knowledge before starting.

General instructions

- **You should choose either set A or B. You will have to answer questions about a paper and solve a related programming exercise.** Aim to finish them, but you're not expected to; we're at least as interested in your thought processes as your actual answers. It is okay if you can only finish a subset of the questions. We are in the first iteration of this PDF so we haven't calibrated yet.
- **You should aim to spend 10-15 hours over the course of at most 10 days.**
- Feel free to email me if you have any questions, both about problems and research.
- You should code in python, using all the packages that you want. For deep learning, preferably use Tensorflow, but PyTorch is also fine.
- You should aim for concise answers, simple yet effective solutions and short, clear code.
- You are encouraged to read all available resources (papers, Stack Overflow). However, you should not ask any person (except me) for help.
- After you have spent 10-15 hours, send me an email with your work.

1 Problem set A: planning under partial observability

Read the paper: QMDP-nets.

For background you can start here.

1.1 Questions: POMDP planning

1. A partially observable policy has to trade off exploration vs. exploitation. In the case of partially observable grid navigation, what is exploration and what is exploitation?
2. QMDP is an algorithm for solving POMDPs. Does it give an optimal solution? Why?
3. How does the QMDP algorithm trade off exploration vs. exploitation?

1.2 Questions: QMDP-net

1. QMDP-nets are trained with backpropagation through time (BPTT). Could we train QMDP-nets for a single step, without BPTT? Why?
2. In the navigation examples QMDP-net receives a map that shows occupied and free cells. What do you think would happen if some of the cells were falsely indicated on the map?
3. (optional) We want to use QMDP-net to learn navigation on a real robot. The robot receives a map of the building. It has differential drive and a LIDAR sensor. What are the most important challenges in your opinion? Try to list three or more. If you have ideas how to address the challenges you can briefly describe them.
4. (optional) Read the paper on TreeQN. What are the positive and negative aspects in relation to QMDP-net?
5. (optional) Can TreeQN address partially observable tasks? How? What is the limitation? Do you have any ideas how to deal with the limitations?

1.3 Exercise: soft-max QMDP-net

1. In its value iteration module, QMDP-net takes the maximum over actions (Eq. 8 of the paper). Understand the source code (available here) and replace the max operation with soft-max.
2. Evaluate if this change is beneficial. You can compare learning curves and the learned policy performance, and whatever else you find interesting. Turn in (1) the modified code, (2) the summary of your findings. Support your findings with results.
3. Full training of the default example should take a few hours on a laptop. You can also terminate training early.

1.4 (optional) Exercise: grid navigation

1. Write a script for solving grid navigation. The problem is similar to the partially observable grid navigation experiments with QMDP-net, except now it is fully observable, i.e. the agent always knows its own state. You will need to specify the MDP model manually. Choose reasonable reward and transition functions.
2. Use value iteration to solve the MDP. You can use an existing implementation of value iteration, or optionally, implement it yourself.
3. You can specify the grid manually, and choose a start and goal state. Optionally, you can generate these at random.
4. Turn in (1) your code (2) a simple visualization of your grid world and the execution of a policy.

2 Problem set B: particle filtering

Read the paper: Particle Filter nets.

For background you can start here or here.

2.1 Questions: particle filters

1. In a particle filter what is the relation between the number of particles and estimation error? How about computational cost?
2. Why do we use resampling in a particle filter? When is resampling important?
3. (optional) We are using particle filtering with a model that has a deterministic transition function. What can go wrong with resampling?
4. (optional) What is the class of probability distributions that we can well approximate with a (1) histogram filter, (2) Kalman filter and a (3) particle filter?

2.2 Questions: PF-nets

1. The localization experiments in the paper assume a map is input to that network. If there were no map, could we still learn a similar localization task? If not, is there a simplified setting that would work?
2. The loss we used to train PF-net is the squared distance between the true pose and the mean particle. What would be an alternative loss function? Can you think of a localization setting where our loss function might not work well?
3. (optional) If we wanted to reproduce the PF-net localization experiments in the real world, what would be the important challenges? If you have ideas how to address the challenges you can briefly describe them.
4. (optional) Are you familiar with deep learning techniques for transferring from simulation to real-world?

2.3 Exercise: robot localization

1. Implement a particle filter as a computational graph (neural network). Use your particle filter to solve the problem below (adapted from CMU 16-899C Assignment 1).
2. You have a robot whose state is x, y, θ. Initially, it is at pose $0, 0, 0$ with no uncertainty. It attempts to move forward at 1 m/s while rotating at 0.1 radian/s for 10 seconds. This command is executed in an open-loop fashion, that is, it does not change its control based on sensor feedback. It has some noise in its motion update (pick any reasonable distribution). Every second it receives a GPS sensor observation; that is, it gets told its x, y position (but not its orientation θ) corrupted by normally-distributed independent noise of standard deviation of 2 meters.
3. Your "neural network" will not have any learnable weights. Instead, it is simply an implementation of particle filtering in Tensorflow (or PyTorch). Note that many numpy operations have their Tensorflow equivalent. The purpose of this exercise is to make yourself familiar with the standard operations in Tensorflow, and not to train a neural network. You only need to implement the particle filter algorithm in Tensorflow, the rest of your code can be in Python.
4. Turn in (1) your code (2) plot the robot's position and uncertainty vs. time during one (or more) sample runs obtained with your implemented code. Your graph may show uncertainty ellipses for the robot in 1-second intervals.
5. (optional) At the end of the 10 seconds, has the robot found out any information about its orientation at time t = 3s? Why or why not?