**Introduction to Reinforcement Learning (RL) in PyTorch**

**Recap of Supervised Learning:**

* So far, we’ve primarily concerned ourselves with supervised learning problems (mostly classification). In supervised learning, we are given some sort of training data consisting of input/output pairs, with the goal being to be able to predict the output given some new inputs after learning the model. For example, we previously looked at a Convolutional Neural Network (CNN) classification model for MNIST; given a training set of 60000 digit images and corresponding digit labels (e.g. ‘5’), we learned a model that was capable of predicting the digit label of new MNIST images. In other words, something like (but not exactly):
* A diagram of a diagram of a layer

  Description automatically generated with medium confidence
  + What if we want to learn how to perform more complex behaviors, where data collection can be expensive? How do you teach a robot to walk? Self-driving cars? How do you defeat human champions in the game of Go?

**Reinforcement Learning**

* Enter Reinforcement Learning. In Reinforcement Learning, our model (commonly referred to as an agent in this context) interacts with an environment by taking actions ‘a’ and receives some sort of feedback from the environment in the form of a reward ‘r’. In this sense, reinforcement learning algorithms learn by experience. We call the trajectory of going from start to finish of a task an ‘episode’, and often our agent will learn by undergoing many episodes.
* A diagram of a cycle

  Description automatically generated
  + Many reinforcement learning algorithms are modeled as Markov Decision Processes (MDPs). In these settings, we have a concept of a state ‘s’, which encapsulates the situation of the agent (e.g. location, velocity). From each state , the agent takes an action , which results in a transition from one state to another state . In many settings, there is stochasticity in this transition, meaning that there is a distribution over conditioned on and . Often, several of these states are considered episode ending, after which the agent can no longer make any transitions or collect any more reward. These correspond to states such as reaching the final goal, a game concluding, or falling of a cliff. In the end, our goal is to learn a policy ‘’, or a mapping from states to actions.
  + In an MDP, we assume that we can always tell which state our agents is in. However, this isn’t always the case. Sometimes, all we have access to are observations that provide information the state , but not enough to precisely pinpoint the exact one. We call such settings Partially Observable Markov Decision Processes (POMDPs). Imagine for example a Roomba being trained to navigate a living room with RL. From its infrared and mechanical “bump” sensors, it receives partial information () as to where it might be, but not a definitive location (. Operating as a POMDP adds a whole layer of complexity to RL algorithms. For the rest of day though, we’ll focus on MDPs, as their much simpler and easier to use to teach basic concepts.

**A simple MDP example**

* In the above example, we can see the 3 possible states fo the agent as , , and , with 2 actions and available from each state. We can see that the each action doesn’t lead to a deterministic transition to the next state, as shown by multiple paths from each action. Note that each of the outcomes of an action are labeled with a small black number between 0 and 1. This denotes the probability of that outcome (which state we end up at) given the action; as these are probabilities, the sum of the probabilities of arriving at each of the next states given a previous state and selected action is 1.
* A diagram of a diagram of a molecule

  Description automatically generated with medium confidence
* **Objective:**
  + The goal of the agent is to maximize the total Reward ‘R’ it can receive over some number of steps. It is important to ensure the reward actually captures the true goal we want the agent to achieve. The agent will dutifully attempt maximize the objective it is given, without any considerations to any implicit objectives that human may desire. There are more than a few (amusing) anecdotes of RL agents learning undesirable behaviors by exploiting some aspect of the reward function. As such, defining this reward requires special care.
  + One countermeasure commonly deployed by RL researchers is the concept of discounted rewards. This is done with a multiplicative term : a reward ‘T’ steps in the future is discounted as . Using discounting encourages the agent to finish the task sooner rather than later, a common implicit criterion. With discounting then, the RL agent’s goal is to maximize:
    - A black and white symbol

      Description automatically generated
  + This is far from the complete solution to making our rewards accurately capture our desired objectives, but achieving higher rewards sooner rather than later is an almost universal preference, so we almost always add it. Designing a good reward function can be an art that is highly dependent of the task.
* **Reinforcement Learning as Supervised Learning?**
  + At first, this doesn’t seem to different from the supervised methods we’ve looked at before, and some natural questions might arise:
    - Why can’t we just treat RL as a supervised task? Why can’t we use the reward (or rather, the negative of the reward) as our supervised loss?
  + Unlike in supervised learning, in reinforcement learning, we often don’t have a pre-apportioned dataset to learn form. In some problems set-ups, we may have examples of other agents (oftentimes humans) performing the desired task, but these aren’t necessarily optimal examples of how to maximize the reward which is what we want to learn. In most RL settings, we don’t have any examples of state-action trajectories beyond what our agent experiences through trial-and-error, which are even more suboptimal

**Open AI Gym:**

* Before we dive any deeper into implementing reinforcement learning models, first we need an environment. Remember, the goal is to learn an agent that can interact with an environment in the way we want, so we need something that our agent can interact with and receive rewards from. In robotics, this is often the real world (or some set-up in the real world). However, it is oftentimes cheaper and quicker to first test our algorithms in simulated settings. There are a number of tasks that are popular benchmarks for the reinforcement learning community, such as ‘cart pole’, ‘mountain car’, or ‘Atari 2600 games’. In the spirit of accelerating progress and promoting openness in the research community, Open AI has very nicely coded up ‘OpenAI Gym’, which has implementations of many of these environments for public use. We will be using these environments, as it allows us to focus on the algorithms themselves, instead of worrying about implementing each problem setting ourselves.