Reinforcement Learning

Bill Paivine and Howie Choset
Introduction to Robotics 16-311

What is ML

Machine learning algorithms build a model based on training data to make predictions or decisions without being explicitly programmed for a particular task.

This can be seen as learning a function which is characterized by a training dataset, which performs well on data points never seen before.

https://en.wikipedia.org/wiki/Machine_learning
https://www.ml.cmu.edu/research/index.html

What is ML

Machine learning algorithms build a model based on training data to make predictions or decisions without being explicitly programmed for a particular task.

This can be seen as learning a function which is characterized by a training dataset, which *hopefully* performs well on data points never seen before.

https://en.wikipedia.org/wiki/Machine_learning
https://www.ml.cmu.edu/research/index.html

Parametric vs. non-parametric learning

Generally, the two types of ML are parametric and non-parametric learning

Parametric vs. non-parametric learning

Generally, the two types of ML are parametric and non-parametric learning

Parametric learning: The function approximator is parameterized by numeric parameters, changes to which change the function that is approximated. Examples include regression, neural networks, etc. Most modern machine learning is focused on parametric learning.

Parametric vs. non-parametric learning

Generally, the two types of ML are parametric and non-parametric learning

Parametric learning: The function approximator is parameterized by numeric parameters, changes to which change the function that is approximated. Examples include regression, neural networks, etc. Most modern machine learning is focused on parametric learning.

Nonparametric learning; Where your model of the approximated function does not have implicit parameters. Examples include decision trees, k-nearest neighbors, etc.

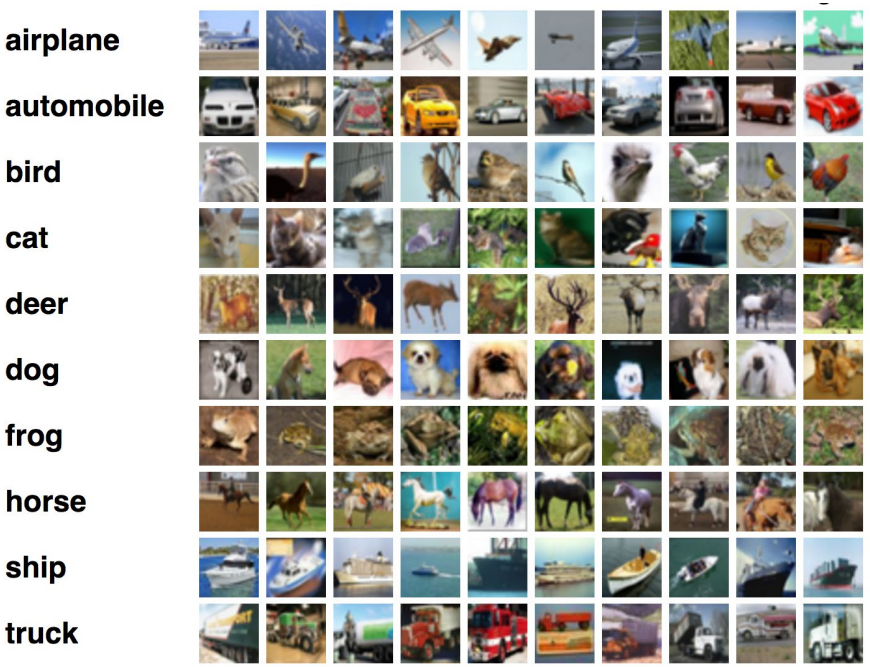
Supervised Learning

Within ML, there are distinctions based on the type of training dataset used.

Supervised learning is learning when the training data includes the ground truth for each datapoint (these are called *labels* for each instance of training data).

Examples include: Image classification, speech recognition, etc.

See https://en.wikipedia.org/wiki/Supervised_learning



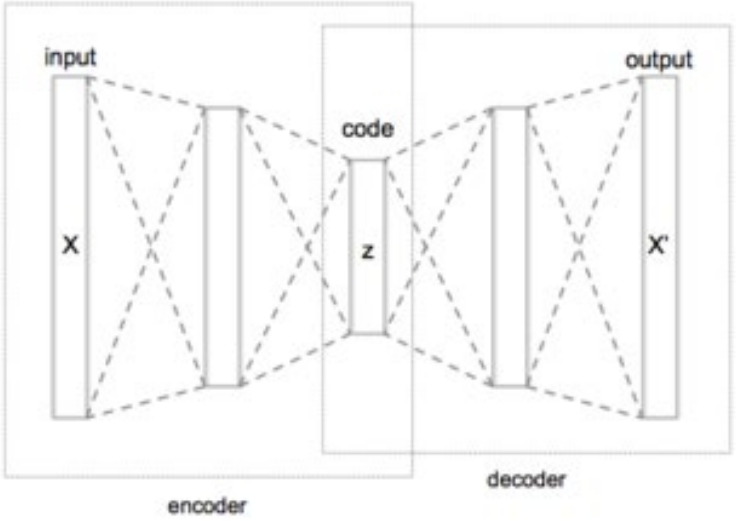
Unsupervised learning

Unlike supervised learning, no labels are included with the training data.

Since there are no labels, the model must find patterns in the training dataset purely from the examples.

A common example is the autoencoder.

Autoencoders try to “compress” an input to a smaller encoding, learning the “important” distinguishing features of the data.

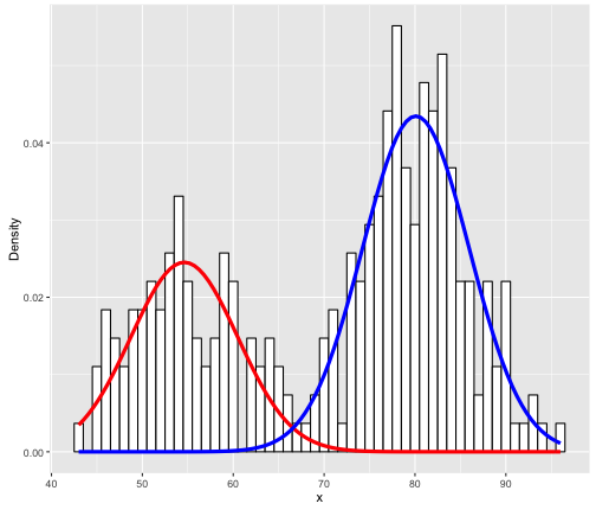


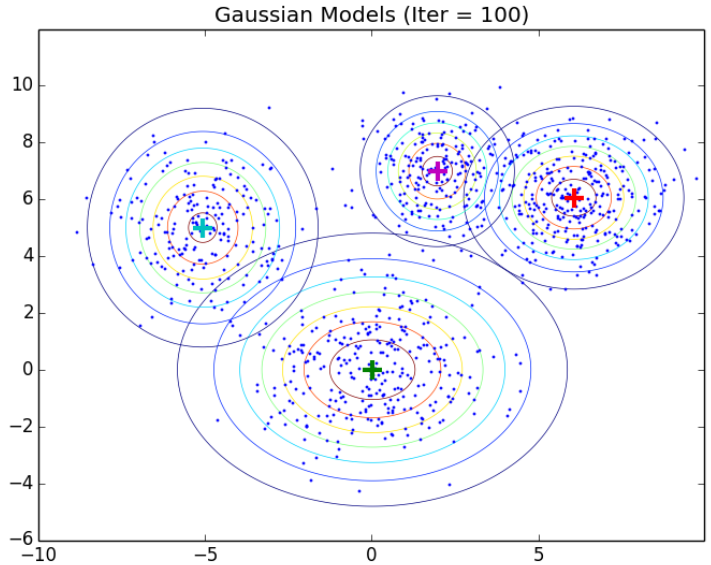
*a form of unsupervised learning

Gaussian Mixture Models

Mixture model: probabilistic model about a subpopulation without having additional information about the subpopulation

Example: housing cost estimation based on location without knowing information about neighborhoods, etc.





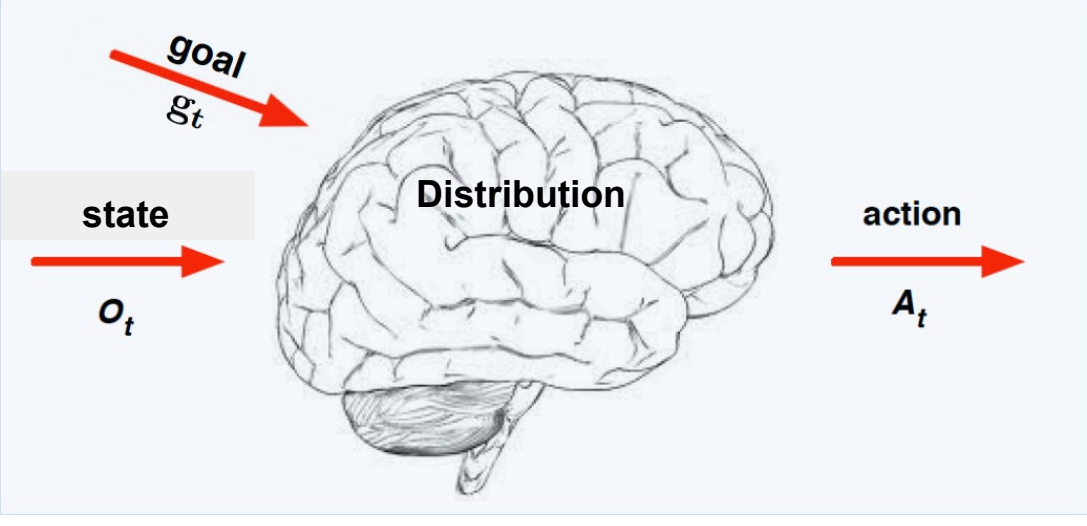
Reinforcement learning

Reinforcement learning can be viewed as somewhere in between unsupervised and supervised learning, with regards to the data given with training data.

Reinforcement learning is more structured, with the goal of training some “agent” to act in an environment.

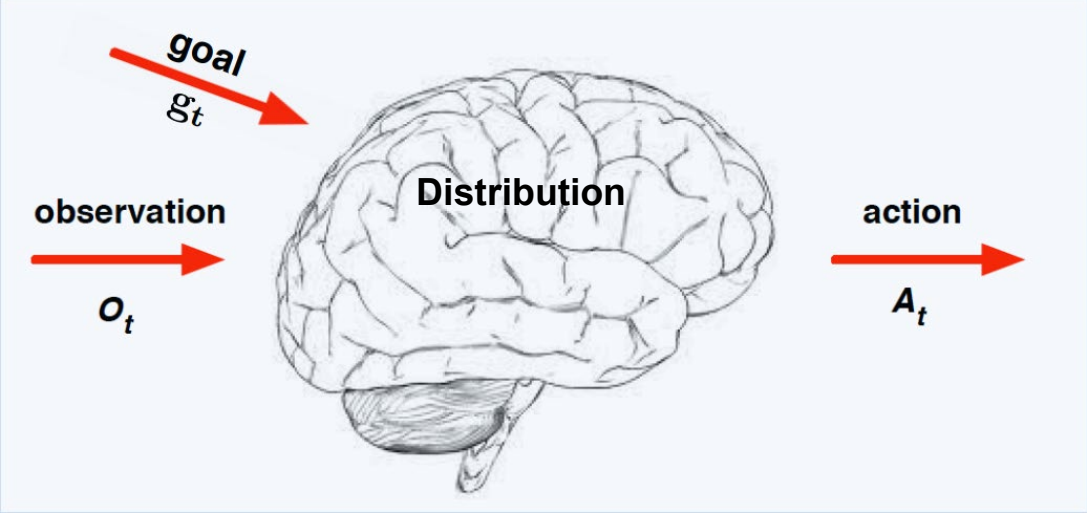
What is RL

Generally speaking, RL is training some “agent” to map sequences of “observations” (of the environment) to actions, for the purpose of achieving a particular goal. We represent the agent’s **policy** as a function, which takes a state as an input, and outputs a probability distribution over the possible actions as an output.



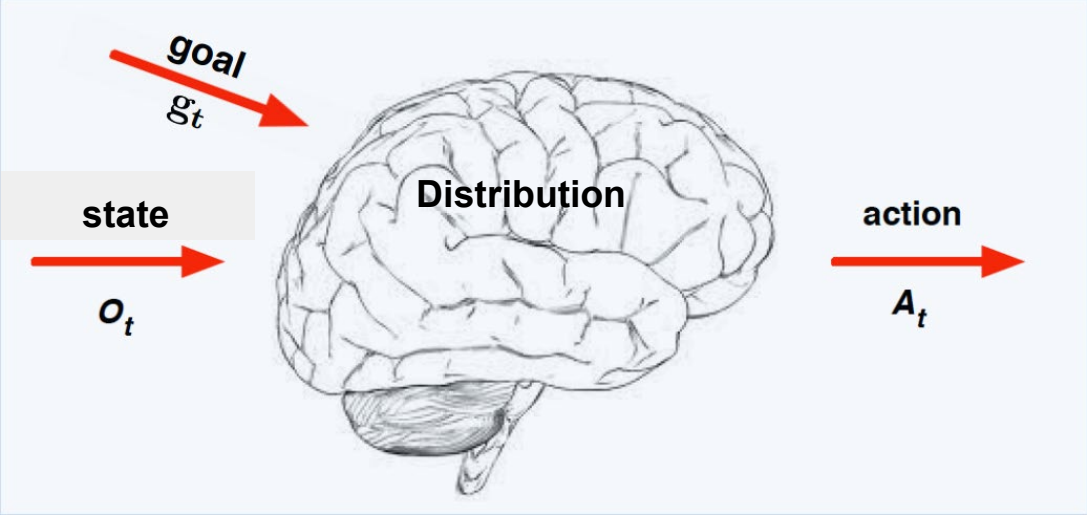
What is RL

Generally speaking, RL is training some “agent” to map sequences of “observations” (of the environment) to actions, for the purpose of achieving a particular goal. We represent the agent’s **policy** as a function, which takes a state as an input, and outputs a probability distribution over the possible actions as an output.



What is RL

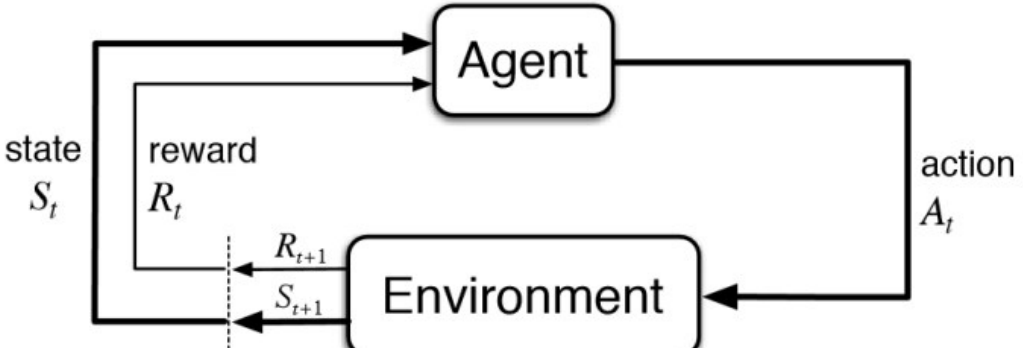
Generally speaking, RL is training some “agent” to map sequences of “observations” (of the environment) to actions, for the purpose of achieving a particular goal. We represent the agent’s **policy** as a function, which takes a state as an input, and outputs a probability distribution over the possible actions as an output.



Reinforcement learning setup

The goal is characterized by some **reward**, which is given to the agent by the environment, signalling when the agent achieves the goal. You want to the reward to accurately reflect what you want. Note the the goal, say self-balance the robot, may not be well-represented by the reward.

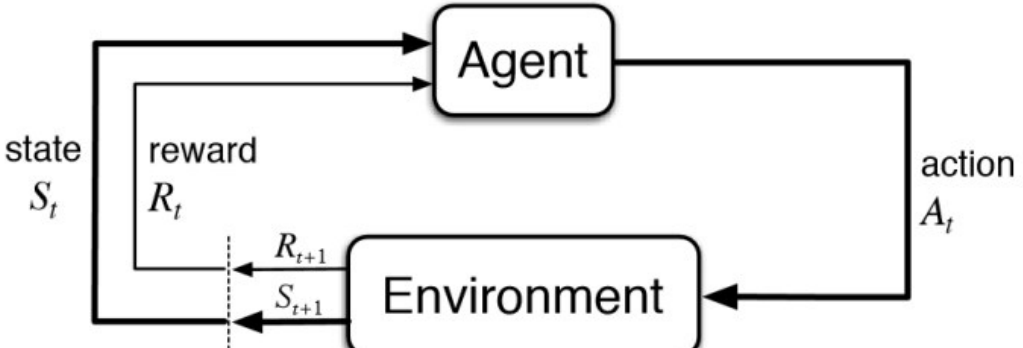
agent learns which *action* to select given the current *state*, with the purpose of maximizing the long-term reward.



Reinforcement learning setup

The goal is characterized by some **reward**, which is given to the agent by the environment, signalling when the agent achieves the goal. You want to the reward to accurately reflect what you want. Note the the goal, say self-balance the robot, may not be well-represented by the reward.

agent (typically represented by a neural network) learns which *action* to select given the current *state*, with the purpose of maximizing the long-term reward.



Loss =
-(Reward - E[Reward])
p(A)

Updating the Policy

Remember that the neural net update equation:

$$\theta \leftarrow \theta - \alpha \nabla_\theta L$$

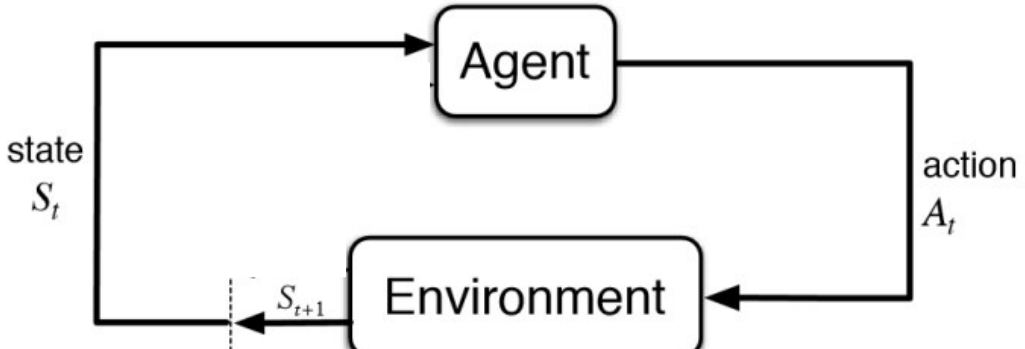
Also, remember our loss function mentioned earlier:

$$L(\theta, s, a) = -(R - E[R])(\pi(\theta, s)[a])$$

Reinforcement learning execution

The goal is characterized by some **reward**, which is given to the agent by the environment, signalling when the agent achieves the goal. You want to the reward to accurately reflect what you want. Note the the goal, say self-balance the robot, may not be well-represented by the reward.

agent (typically represented by a neural network) learns which *action* to select given the current *state*, with the purpose of maximizing the long-term reward.



Episodic RL

Episode - a sequence of observations, rewards, and corresponding actions which start a designated start state and terminate at an ending state as determined by environment.

Episodic RL

Episode - a sequence of observations, rewards, and corresponding actions which start a designated start state and terminate at an ending state as determined by environment.

Environment is the environment, start, goal, reward, state transition

Episodic RL

Episode - a sequence of observations, rewards, and corresponding actions which start a designated start state and terminate at an ending state as determined by environment.

In other words, the agent interacts with the environment through an “episode”, where the agent starts at some initial state, and attempts to maximize reward by picking the best action after each state transition, until a terminal state is reached.

Episodic RL

Episode - a sequence of observations, rewards, and corresponding actions which start a designated start state and terminate at an ending state as determined by environment.

In other words, the agent interacts with the environment through an “episode”, where the agent starts at some initial state, and attempts to maximize reward by picking the best action after each state transition, until a terminal state is reached.

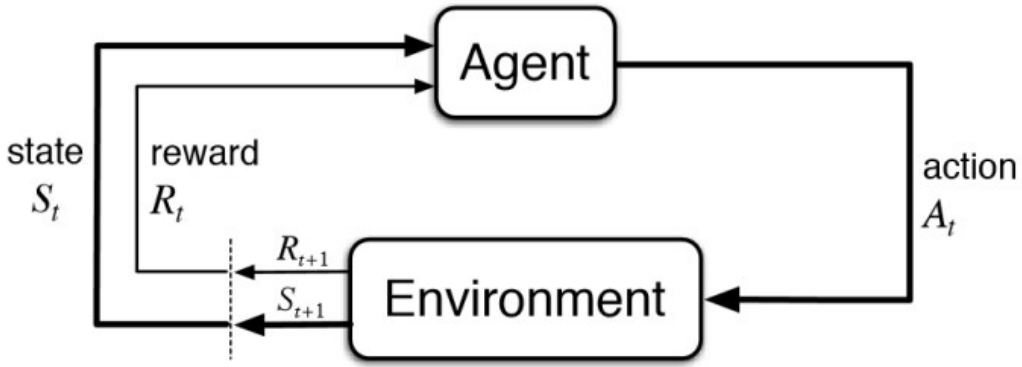
An example of an episodic RL task would be a robot which balances a pole in the upright position--it receives reward for each second the pole is upright, but the episode ends when the pole falls over (or when it times out). (HW 4, Lab 4!!)

Time Discrete RL

In an episode, time is broken up into discrete time steps.

At each time step, the environment provides the current state and a reward signal.

The agent can then choose an action, which affects the environment in some way, and the process repeats.



Episode

Single episode of n time steps

$(s_1, a_1, r_1), (s_2, a_2, r_2),....(s_n, a_n, r_n)$

Each tuple contains a state, an action made by the agent, and the reward given by the environment immediately after the action was made.

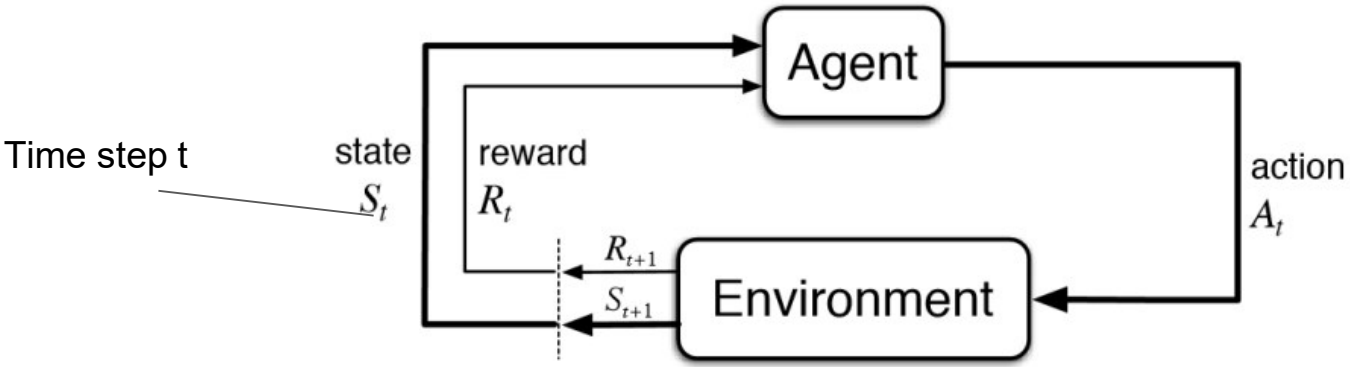
The episode starts at a **starting state** and finishes in a **terminal state**.

Time Discrete RL

In an episode, time is broken up into discrete time steps (not continuous).

At each time step, the environment provides the current state and a reward signal.

The agent can then choose an action, which affects the environment in some way, and the process repeats.



Both learning and execution

Rewards: Challenge in Interpreting Reward Signal

1. (temporal) The reward for a good action or set of actions may not be received until well into the future, after the action is given (long-term reward)
2. (sparse) The reward may be sparse (e.g. receiving a reward of 1 for hitting a bullseye with a dart, but a reward of 0 for hitting anything else)
3. (quality) Reward may not guide the agent on each step of the way.

So, in learning, we must be careful not to make many assumptions about the reward function.

Interpreting Rewards

However, there is one assumption that is made:

A given reward is the result of actions made earlier in time

In other words, when we receive a reward, any of the actions we have chosen up to this point could have been responsible for our receiving of that reward.

In addition, we often assume that the closer to an action we receive a reward, the more responsible that action was.

Episode (reminder)

Single episode of n time steps

$(s_1, a_1, r_1), (s_2, a_2, r_2),....(s_n, a_n, r_n)$

Each tuple contains a state, an action made by the agent, and the reward given by the environment immediately after the action was made.

The episode starts at a **starting state** and finishes in a **terminal state**.

Interpreting Reward: Discounted Rewards

These assumptions gives rise to the idea of the **discounted reward**.
With the discounted reward, we choose a **discount factor (γ)**, which tells how correlated in time rewards are to actions.

$$g_t = \sum_{t=T}^{n} \gamma^{t-T} r_t$$

As you can see, at time **T**, the discounted reward is the decaying sum of the rewards which are received after the current time step until the end of the episode.

Episode with Discounted Rewards (roll out)

Single episode of n time steps

$(s_1, a_1, g_1), (s_2, a_2, g_2), ...(s_n, a_n, r_n)$

Calculating the discounted rewards can easily be done in reverse-order, starting from the terminal state and calculating the discounted reward at each step using the discounted reward for the next step to simplify the sum.

$$
\begin{aligned}
g_1 &= r_1 + \gamma g_2 \\
g_2 &= r_2 + \gamma g_3 \\
g_3 &= r_3 + \gamma g_4 \\
&\vdots \\
g_{n-1} &= r_{n-1} + \gamma g_n \\
g_n &= r_n
\end{aligned}
$$

Imitation Learning: Circumventing Reward Function difficulties

Tasks may be interpreted as a reinforcement learning problem can often be solved reasonably well with **Imitation Learning**

Imitation learning is a supervised learning technique where an agent is trained to mimic the actions of an **expert** as closely as possible.

Why bother training an agent if we already have an expert?

Imitation Learning: Circumventing Reward Function difficulties

Tasks may be interpreted as a reinforcement learning problem can often be solved reasonably well with **Imitation Learning**

Imitation learning is a supervised learning technique where an agent is trained to mimic the actions of an **expert** as closely as possible.

Why bother training an agent if we already have an expert?

The expert may be expensive to use, or perhaps there is only one expert, or we need to query the expert more often than it can respond.

In general, imitation learning is useful when it is easier for the expert to demonstrate the desired behavior than it is to directly create the policy or create a suitable reward function.

Imitation Learning + Reinforcement Learning

In the case we have an expert whose performance we want to exceed, and we can create a good reward-function, we can get benefits from both imitation learning and reinforcement learning.

This can stabilize learning, and allow the agent to learn an optimal policy much quicker.

For example, a policy for the self-balancing robot lab can be learned with imitation learning, and then further trained with reinforcement learning.

Imitation Learning Example: Self-Piloting Drone

First, a human drives a drone, recording the video from the drone's camera and the corresponding human commands.

Then, a neural network is trained, such that when given a frame, it outputs the action the human would have performed.

Now the drone can autonomously navigate like a human!

Imitation Learning Example: Self-Piloting Drone

Imitation learning: Caveats

However, imitation learning is not perfect.

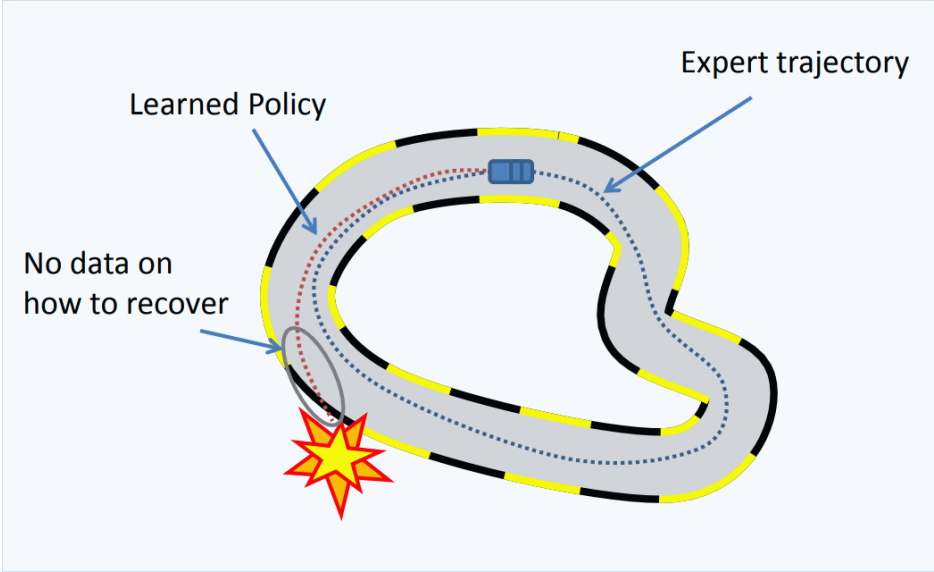
1. the agent cannot exceed the performance of the expert (i.e. it can not improve better than a human)
2. it requires a significant amount of training examples, which may be difficult to acquire (suppose there is only 1 human expert which can do the task)
3. it does not learn how to act in situations for which the expert never demonstrated.
4. the agent does not easily learn how to account for error accumulation from small differences in the learned policy vs expert policy

Imitation learning: Error accumulation

Suppose we train a self-driving car using imitation learning.

In the expert dataset, the expert would never intentionally drive off the side of the road. (does not make a mistake)

Catch 22?



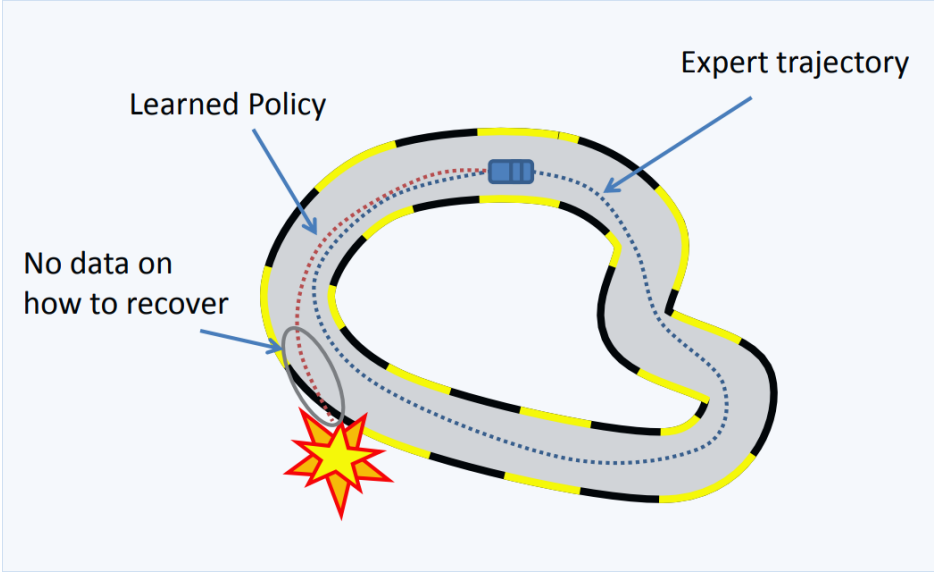
Imitation learning: Error accumulation

Suppose we train a self-driving car using imitation learning.

In the expert dataset, the expert would never intentionally drive off the side of the road. (does not make a mistake)

However, there will be small differences in the learned policy, causing the car's trajectory to drift.

As the car drifts, the more its input differs from training data, causing a buildup of policy mismatch (approximation gets worse), until disaster



Alternative Approach: Policy-based RL

So, while imitation learning can learn a decent policy, we would ideally like to be able to improve the learned policy, or possibly even learn a policy with no expert demonstrations.

However, to do this, we need to know how to **interpret the reward signal,** and we need to know how to update our policy.

To perform policy improvement, we need to have a **parameterizable** policy, such as a neural network.

Evaluating Action Performance

We can judge whether an action was “good” or “bad” by comparing the discounted reward to some **baseline**. (Don’t worry about how we get the baseline yet).

Remember that the *policy* is a mapping from state to a probability distribution over the possible actions.

If the discounted reward was higher than the baseline, then we want to **increase** the probability of the action being selected (for that state).

If the discounted reward was higher than the baseline, then we want to **decrease** the probability of the action being selected (for that state).

Updating the Policy

Remember that the neural net update equation:

$$\theta \leftarrow \theta - \alpha \nabla_\theta L$$

Also, remember our loss function mentioned earlier:

$$L(\theta, s, a) = -(R - E[R])(\pi(\theta, s)[a]])$$

We will replace the R term with the discounted reward, and the E[R] with our baseline, giving the following loss function:

$$L(\theta, s, a) = -(G_t - B)(\pi(\theta, s)[a]))$$

Updating the Policy

Remember that the neural net update equation:

$$\theta \leftarrow \theta - \alpha \nabla_\theta L$$

We will replace the R term with the discounted reward, and the E[R] with our baseline, giving the following loss function:

$$L(\theta, s, a) = -(G_t - B)(\pi(\theta, s)[a]))$$

We can combine these to create a new update equation:

$$\theta \leftarrow \theta - \alpha(-\nabla_\theta((G_t - B)(\pi(\theta, s)[a])))$$

Since neither the rollout or the baseline depend on θ, we can rewrite this equation:

$$\theta \leftarrow \theta + \alpha((G_t - B)\nabla_\theta(\pi(\theta, s)[a]))$$

Updating the Policy

If we let π(θ,s) be the policy, G_t is the discounted return for state *s*, *B* is the baseline, *a*, is the action which was chosen, and θ is the parameters of the policy, we can update the policy as follows:

$$\theta \leftarrow \theta + \alpha((G_t - B)\nabla_\theta(\pi(\theta, s)[a]))$$

Intuition: If the action was “good”, we increase the probability of that action being picked in similar situations. If it were “bad”, we decrease the probability, which will in turn increase the probabilities of all other actions being chosen in similar situations.

Further Readings

One presentation is barely enough to chip away at all there is to learn in RL and ML (There are entire courses, even degrees, which go more in-depth).

Not mentioned in this presentation, but are common in RL is Q-learning, Actor-Critic methods, and more.

Additionally, there are several topics not discussed which can be very important for RL tasks. The most important thing is the idea of exploration vs. exploitation (When learning a task, when should the agent stop trying to figure out which actions are good actions, and start exploiting what it knows? [Problem of local minima]).

There are also methods which use models of the environment, allow for continuous action space, and more.