# **Reinforcement Learning 1:**

Components:

* Environment
* Reward Signal
* Agent:
  + Agent state
  + Policy
  + Value function (probably)
  + Model (optionally)

Agent’s goal is to maximize the cumulative reward (return) G\_t = \sum\_{t} R\_{t+1}, generally considered finite. Can be sparse (i.e. many R\_{t+1} give 0). Reward are scalar, can be positive or negative

Expected cumulative reward from a state s is the value v(s).

A mapping from states to actions is called policy

If a value is a fn of state and action it is notated q(s, a)

A decision process is Markov if “The future is independent of the past given the present” (once the state is known the history can be discarded)

Partial observable Markov decision processes:

E.g. “robot with camera”

(Can still be Markov environment)

To deal with partial observability, agent can use

1. Last observation (though this may be insufficient)
2. Entire history (often too expensive)
3. S\_t = f(S\_{t-1}, Observation\_t), as in RNN

Policy maps agents state to action:

1. Deterministic A = \pi(S)
2. Stochatic \pi(A | S) = p(A | S)

V\_{pi} = E [ R\_{t + 1} + \gamma R\_{t + 2} + \gamma^2 R\_{t + 3} … ]

\gamma discounts value of less certain future actions

Agents:

Value Based (Value fn no explicit policy)

Policy Based (Policy no explicit value fn)

Actor Critic (Value fn and Policy)

Any of above can also be Model Free or Model Based, depending on whether or not they have a model

Models can sometimes lead to peculiar solution based on incorrect modelling.

Prediction: evaluate the future (for a given policy)

Control: optimize the future (find best policy)

# 

# **Reinforcement Learning 5: Function Approximation and Deep Reinforcement Learning**

Step 1: Value Function Approximation

As a following robot, our loss is not actually well defined. If our loss were merely a function of our final position after X amount of time relative to the position of the leader car, function approximation would be more important. We would want to build a value function to approximate this far off value for our training. But in our case, the goal is to follow along the same path, insofar as it is possible, so our true loss can be essentially determined by some function of distance and velocity vectors.

If we could combine those features to get a good loss function, we would not need to work out a function approximation. Perhaps combining these is not entirely obvious, but we might be able to craft a function that is “good enough” by hand and thus skip the following section.

Function Approximators:

* Neural Network
* Decision Tree (more rigid boundaries)
* KNN (Simple but reasonable)
* Simpler functions, e.g. linear
* Coarse Coating (bad at high dimensional spaces, usually binary features with thresholds)

Though non-Neural Networks require feature extraction, which is not so easy for our use case if we intend to use the raw image data

Alternatively this might be a good opportunity to bring in features that we can know during training time, like angles and velocities of the cars and distance between the cars (this could be calculated with a neural network, but should probably instead be done using the tags on the cars), and use a linear combination of these features to build the value approximation function.

Regardless, simple SGD on the approximator won’t work if we don’t have a true value of our loss, hence we need to approximate it: using MC (Monte Carlo) or TD (Temporal Difference).

Monte Carlo provides an unbiased sample but is more taxing to collect training data for.

TD provides a biased sample with bootstrapping, basing our “loss” for our future prediction on a future (presumed better) future prediction rather than the true outcome.

-- 57 minutes