Reinforcement learning is about discovering data/labels through exploration and a reward signal, but starting off without data nor labels (zero experience).

The process is: agent observes the environment’s state s, then chooses an action a and receives a reward r and a new state s\_ from the environment.

State variables:

* X coordinates
* Y coordinates
* X velocity
* Y velocity
* Angle
* Angular velocity
* Left leg grounded
* Right leg grounded

Actions:

* 0: do nothing
* 1: fire left
* 2: fire down
* 3: fire right

Rewards:

* increased/decreased the closer/further the lander is to the landing pad.
* increased/decreased the slower/faster the lander is moving.
* decreased the more the lander is tilted (angle not horizontal).
* increased by 100 points for each leg that is in contact with the ground \*changed\*
* decreased by 0.03 points each frame a side engine is firing.
* decreased by 0.3 points each frame the main engine is firing.
* Increased by 100 points for landing safely
* Decreased by 100 points for crashing

An episode is considered a solution if it scores at least 200 points.

What we need to do is to learn a policy or model that will maximise expected rewards. Policy pi(a|s) is the probability to take action a at state s. We will use neural network for this.

**Policy Gradient**

Policy gradient methods directly learns a policy π, a strategy for selecting actions in an environment, without explicitly computing value functions. The policy is a rule for actions. For each state s, the policy defines a probability distribution over the set of possible actions. Mathematically, this can be expressed as π(a∣s), representing the probability of taking action a in state s.

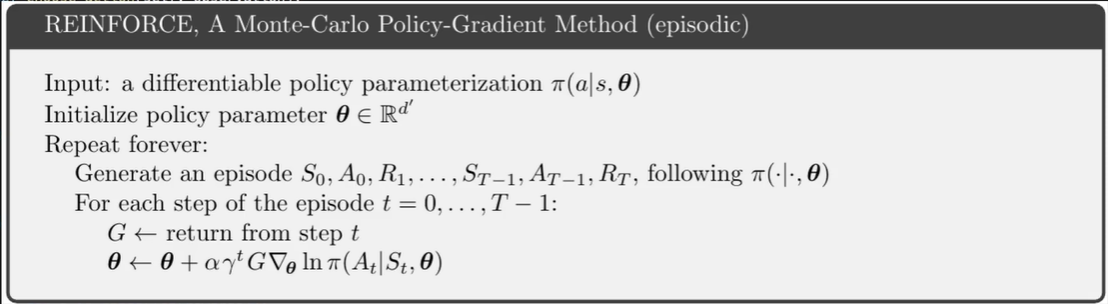
The key idea behind policy gradient methods is to parameterize the policy and then optimize these parameters to maximize the expected cumulative reward. We will be using deep neural networks to approximate the probability distribution π.

This is a Monte Carlo method, ie it doesn’t learn at each time step (like a Temporal Difference learning method), but it learns at the end of every episode. For my implementation, the learning will be done after every episode (as opposed to a batch of episodes).

**Some information about the environment**

* Neural network implemented using Tensorflow
* Activation function: relu
* Optimizer used for neural network is Adam

Hyperparams:

* Alpha (learning rate) = 0.0002
* Gamma (discount factor) = 0.99
* Layer 1 size = 256
* Layer 2 size = 256
* Episodes = 5000
* 

References:

https://www.youtube.com/watch?v=mA9rxgOQyE4&t=10s (my code is 99.9999% from here)

Hyperparam tuning:

* Learning rate 0.0003, 0.0005, 0.0001
  + The learning rate of the Adam optimizer
* Gamma 0.99, 0.999, 0.9
  + The discount factor for future rewards
* # neurons in fully connected layer 256,128,512
  + The number of neurons in the fully connected layers of the MLP (2 layers)
* time\_threshold 40,30,20
  + A new variable introduced to discourage the rover from hovering (which 1. is time costly, 2. Goes counter to the objective)
  + After a time equivalent to time\_threshold in seconds from the start of the episode, the done variable is forcefully set to True (terminating the episode) and an additional 50 points are deducted from the reward variable
* weight\_decay None, 0.0001, 0.001, 0.01
  + A hyperparameter for the Adam optimizer that regulates r1 regularization (prevents overfitting)

~~Saving and loading the model~~

GAE? DAMN STRAIGHT

Batch training? NAH