<https://medium.com/emergent-future/simple-reinforcement-learning-with-tensorflow-part-0-q-learning-with-tables-and-neural-networks-d195264329d0>

READ THIS HERE FIRST <http://neuro.cs.ut.ee/demystifying-deep-reinforcement-learning/>

* First of all: RL.Typical aspects of a task that make it an RL problem are the following:
* Different actions yield different rewards. For example, when looking for treasure in a maze, going left may lead to the treasure, whereas going right may lead to a pit of snakes. | Rewards are delayed over time. This just means that even if going left in the above example is the right things to do, we may not know it till later in the maze. | Reward for an action is conditional on the state of the environment. Continuing the maze example, going left may be ideal at a certain fork in the path, but not at others.
* Policy-based vs Q-learning. All we need to focus on is learning which rewards we get for each of the possible actions, and ensuring we chose the optimal ones. In the context of RL lingo, this is called learning a policy. We are going to be using a method called policy gradients, where our simple neural network learns a policy for picking actions by adjusting it’s weights through gradient descent using feedback from the environment. There is another approach to reinforcement learning where agents learn value functions. In those approaches, instead of learning the optimal action in a given state, the agent learns to predict how good a given state or action will be for the agent to be in.
* Easiest version is a simple Q-table
* Unlike policy gradient methods, which attempt to learn functions which directly map an observation to an action, Q-Learning attempts to learn the value of being in a given state, and taking a specific action there
* Uses Gym/Universe
* In it’s simplest implementation, Q-Learning is a table of values for every state (row) and action (column) possible in the environment. Within each cell of the table, we learn a value for how good it is to take a given action within a given state.
* Bellmann equation: states that the expected long-term reward for a given action is equal to the immediate reward from the current action combined with the expected reward from the best future action taken at the following state. <formula, and in code its Q[s,a] = Q[s,a] + lr\*(r + y\*np.max(Q[s1,:]) - Q[s,a]), dazuschreiben warum das eine APPROXIMATION dieser ist.. Und ist das nicht bei nur positiven rewards zu hoch?>
* When using ANNs instead of a table: The method of updating is a little different as well. Instead of directly updating our table, with a network we will be using backpropagation and a loss function. Our loss function will be sum-of-squares loss, where the difference between the current predicted Q-values, and the “target” value is computed and the gradients passed through the network. In this case, our Q-target for the chosen action is the equivalent to the Q-value computed in equation 1 above.
* Environments which pose the full problem to an agent are referred to as Markov Decision Processes (MDPs). These environments not only provide rewards and state transitions given actions, but those rewards are also condition on the state of the environment and the action the agent takes within that state. These dynamics are also temporal, and can be delayed over time. To be a little more formal, we can define a Markov Decision Process as follows. An MDP consists of a set of all possible states S from which our agent at any time will experience s. A set of all possible actions A from which our agent at any time will take action a. Given a state action pair (s, a), the transition probability to a new state s’ is defined by T(s, a), and the reward r is given by R(s, a). As such, at any time in an MDP, an agent is given a state s, takes action a, and receives new state s’ and reward r.
* Reward may be delayed, and we don’t see the entire game state (only observations)
* What is a model and why would we want to use one? In this case, a model is going to be a neural network that attempts to learn the dynamics of the real environment. For example, in the CartPole we would like a model to be able to predict the next position of the Cart given the previous position and an action.By learning an accurate model, we can train our agent using the model rather than requiring to use the real environment every time
* Model-based RL: we are going to be using a neural network that will learn the transition dynamics between a previous observation and action, and the expected new observation, reward, and done state. Our training procedure will involve switching between training our model using the real environment, and training our agent’s policy using the model environment. ACTOR-CRITIC MODEL BASED
* Environments which present themselves in a limited way to the agent are referred to as Partially Observable Markov Decision Processes (POMDPs).
* Within the context of Reinforcement Learning, there are a number of possible ways to accomplish this temporal integration. The solution taken by DeepMind in their original paper on Deep Q-Networks was to stack the frames from the Atari simulator. Instead of feeding the network a single frame at a time, they used an external frame buffer which kept the last four frames of the game in memory and fed this to the neural network. This approach worked relatively well for the simple games they employed, but it isn’t ideal for a number of reasons. The first is that it isn’t necessarily biologically plausible. When light hits our retinas, it does it at a single moment. There is no way for light to be stored up and passed all at once to an eye. Secondly, by using blocks of 4 frames as their state, the experience buffer used needed to be much larger to accommodate the larger stored states. This makes the training process require a larger amount of potentially unnecessary memory. Lastly, we may simply need to keep things in mind that happened much earlier than would be feasible to capture with stacking frames. Sometimes an event hundreds of frames earlier might be essential to deciding what to do at the current moment. We need a way for our agent to keep events in mind more robustly.