# Capstone notes

## Bandits vs MDP -

**Bandits**

The agent is presented with the same situation and each time and the same action is always optimal.

**MDP**

Actions influence immediate rewards as well as future states and through those, future rewards.

So how can we represent the dynamics of this interaction? As in bandits, the outcomes are stochastic and so we use the language of probabilities. When the agent takes an action in a state, there are many possible next states and rewards. The transition dynamics function P, formalizes this notion. Given a state S and action a, p tells us the joint probability of next state S prime and reward are

**Episodic task**

Example tetris game

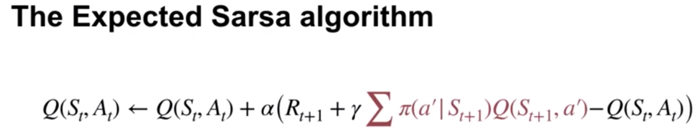
**Continuous task**

Servers scheduling jobs based on their priorities.

Episodic tasks break naturally into independent episodes. Continuing tasks are assumed to continue indefinitely.

**Expected SARSA**

expected Sarsa algorithm explicitly computes the expectation under its policy, which is more expensive than sampling but has lower variance.

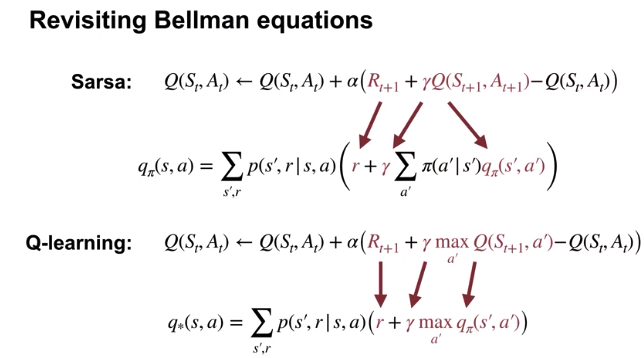
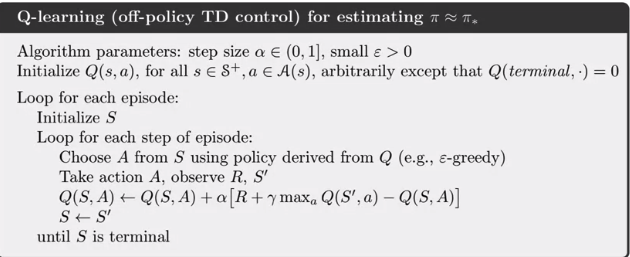


**Q-learning**

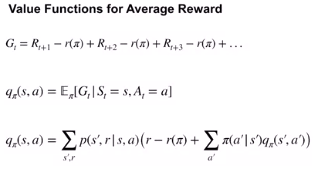
Sarsa is policy iteration

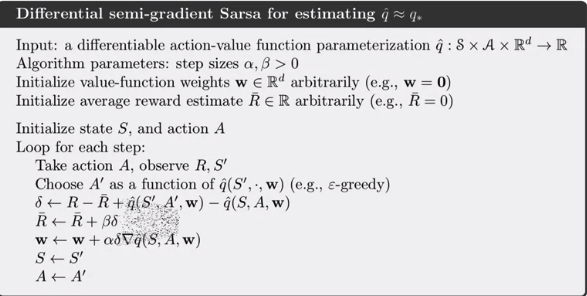
Q-learning is value iteration

Sarsa is sample-based version of policy iteration which uses Bellman equations for action values, that each depend on a fixed policy. Q-learning is a sample-based version of value iteration which iteratively applies the Bellman optimality equation. Applying the Bellman's Optimality Equation strictly improves the value function, unless it is already optimal. So value iteration continually improves as value function estimate, which eventually converges to the optimal solution. For the same reason, Q-learning also converges to the optimal value function as long as the aging continues to explore and samples all areas of the state action space.



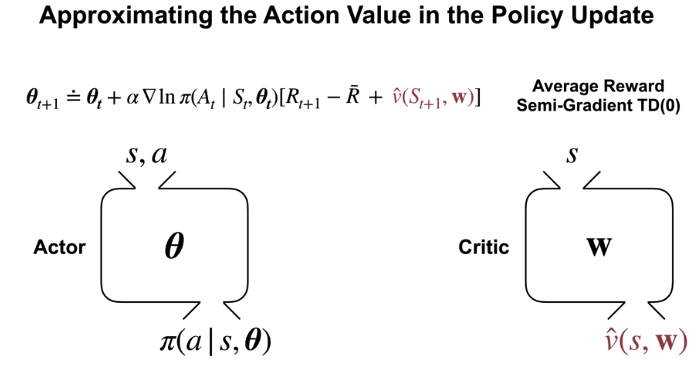
**Average Reward – New way of formulating control problems**

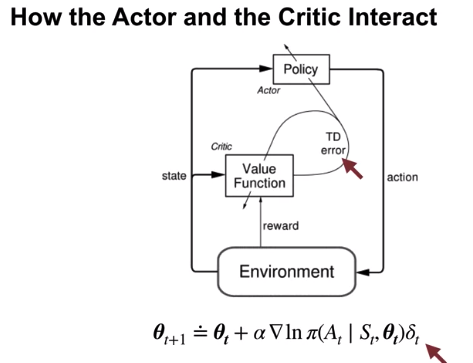


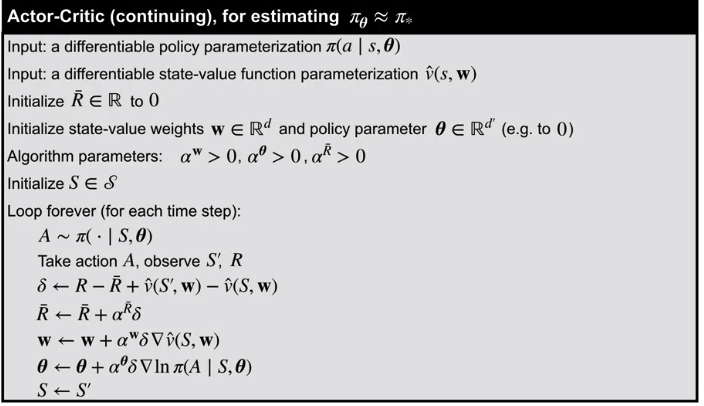


Actor – Critic algorithm

the parameterized policy plays the role of an actor, while the value function plays the role of a critic, evaluating the actions selected by the actor. These so-called actor-critic methods, were some of the earliest TD-based methods introduced in reinforcement learning.







1. Which of the following algorithms are appropriate in a control setting in which updates will be made at every time step?

* Q-learning - Q-Learning uses temporal difference learning updates that are done at every time step with (state, action, next state, reward) transition tuples where the target is the sum of the reward and the max over the action values at the next state.
* SARSA - Expected SARSA uses temporal difference learning updates that are done at every time step with a (state, action, next state, reward) transition tuples where the target is the sum of the reward and the expected action value of the next state.
* Expected SARSA - SARSA uses temporal difference learning updates that are done at every time step with (state, action, next state, reward, next action) transition tuples where the target is the sum of the reward and the action value of the next action at the next state

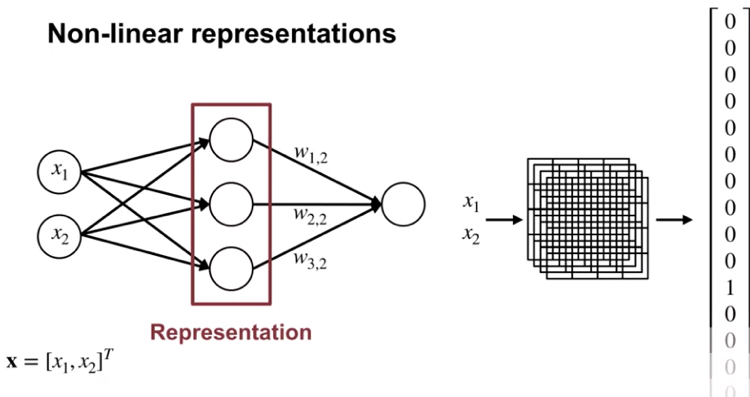
1. Which of the following algorithms are appropriate in a prediction setting in which updates will be made at the end of each episode
   1. Monte-Carlo Prediction - Monte Carlo can be used to estimate the value function with respect to a given policy with experience from the same policy. Thus, it solves a prediction problem. The targets are empirically observed returns by waiting till the end of episodes.
   2. Off policy monte-carlo - Off-Policy Monte Carlo can be used to estimate the value function with respect to a target policy with experience from some behavior policy. The targets are empirically observed returns by waiting till the end of episodes.
2. Which of the following algorithms are appropriate in a tabular setting in which we will be learning a model and using it for planning? [Select all that apply]
   1. Dyna-Q - Dyna-Q uses a model to learn from both simulated and real experience and planning is done by making queries to the model
   2. Dyna-Q+ - Dyna-Q+ uses a model to learn from both simulated and real experience and planning is done by making queries to the model. In addition, Dyna-Q+ can handle non-stationarity in environment well by making use of an exploration bonus to visit long unvisited states and ensure that action-values are up-to-date across the MDP.
3. Which of the following algorithms are appropriate in a control setting in which we are given access to a model?
   1. Policy Iteration- Policy iteration is a method of computing an optimal policy by iteratively finding the value function corresponding to a given policy and then improving that policy. In order to do so, it makes use of the transition probabilities and reward function of the MDP or, equivalently, access to a model. Thus, it is an appropriate algorithm for a control setting with access to a model.
   2. Value Iteration - Value iteration is a method of computing an optimal policy and its value by first finding an optimal value function first and then extracting a policy. In order to do so, it makes use of the transition probabilities and reward function of the MDP or, equivalently, access to a model. Thus, it is an appropriate algorithm for a control setting with access to a model.
   3. Dyna-Q - Dyna-Q can plan by making queries to a model and learn a good policy in that attains large returns. Thus, Dyna-Q is suitable for a control setting with access to a model.
4. Which of the following algorithms are appropriate in a continuing control setting with a discrete action space and function approximation?
   1. Differential softmax actor critic - Differential softmax actor-critic uses function approximation to parameterize a state-conditional categorical distribution over actions. Thus, it is appropriate for a discrete action space setting with function approximation. Differential actor-critic methods are also appropriate for the continuing setting and aim to find a good policy that maximizes average reward.
   2. Differential semi-gradient sarsa - In the average reward setting, differential semi-gradient SARSA finds a near optimal action-value function and hence policy with function approximation. Hence, it is appropriate for a continuing, control setting with function approximation. With linear function approximation with action-values being learned for all the discrete actions, differential semi-gradient SARSA is also appropriate for discrete-action spaces.
5. Which of the following algorithms are appropriate in an online prediction setting with linear function approximation
   1. Semi-Gradient TD - Semi-gradient TD can use linear function approximation and temporal difference learning style updates at every time step.

**Agent Architecture**

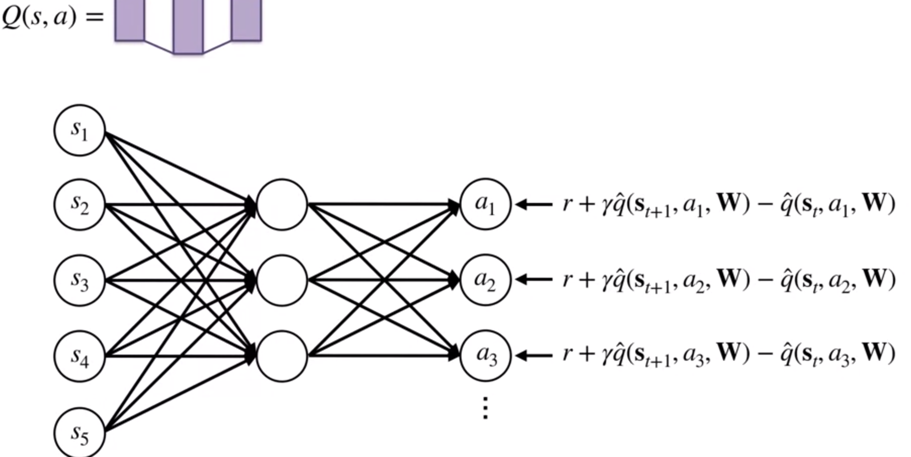
Function Approximator – Tile Coding (huge 10^8features), so we will go with Neural Network. Activation function relu. How to train the NN – No Adagrad – decays stepsizes to 0, RMSProp uses info about curvature of the loss to improve descent step, use Adam (combines curvature info from RMSProp and momentum)

which expiration method we will use – Softmax , This choice could be better because the probability of selecting an action is proportional to the value of that action. This way we are less likely to explore actions that we think are really bad.

**Non-linear Approximation with Neural Networks**

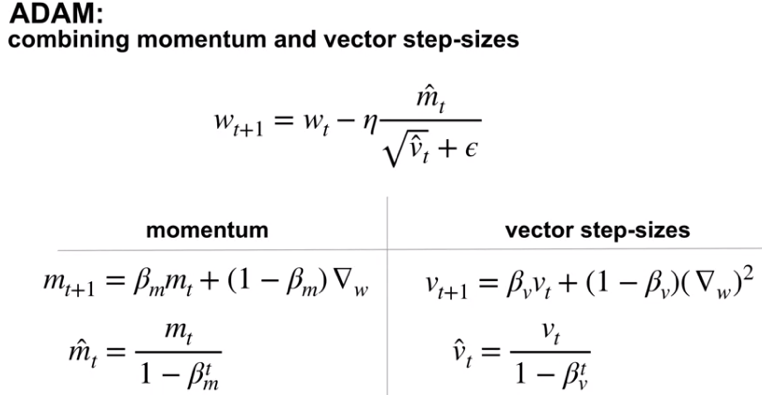


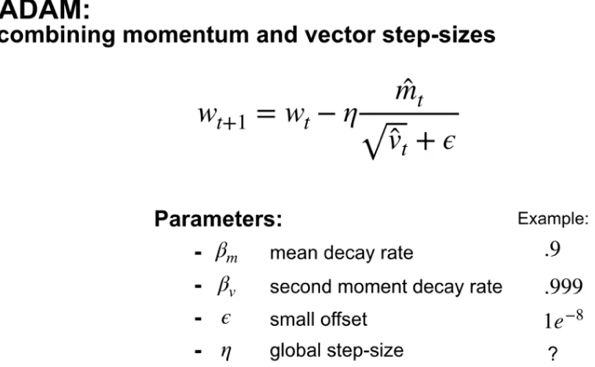
**Agent Details**



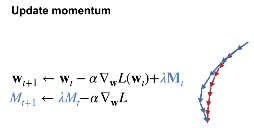
Reduce TD error, updated values for action taken

Adam algorithm – Use momentum and vector step size

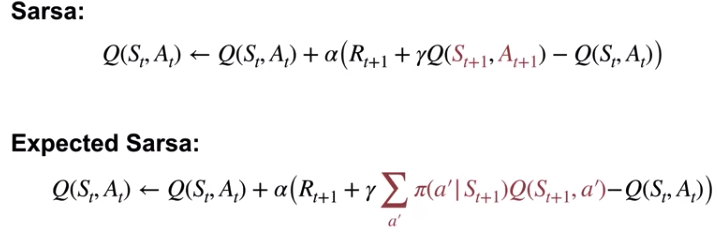


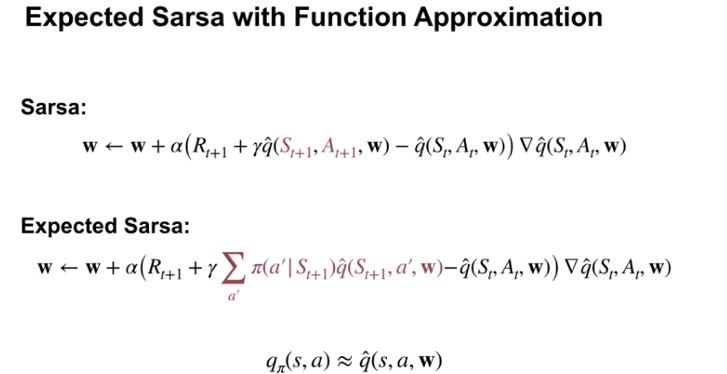


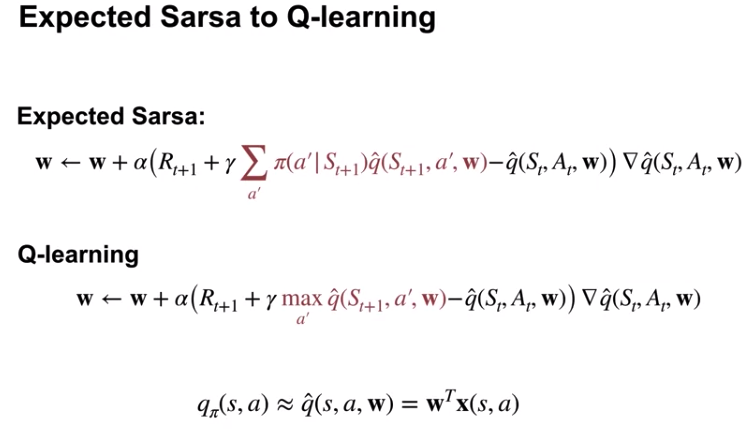
**Optimization strategies for NN**

1. Initialization of NNs – Randomly sample initial weight from a normal distribution with small variance. Scale the variance of the weights by one over the sq. root of the number of inputs.
2. Improve weights by SGD, Momentum and vector size adaptation
3. 
4. Adapting the step sizes for each weight, based on statistics about the learning process in practice results in much better performance. Now, how does the update change? The change is very simple. Instead of updating with a scalar Alpha, there's a vector of step sizes indexed by t to indicate that it can change on each time-step. Each dimension of the gradient, is scaled by its corresponding step size instead of the global step size. There are a variety of methods to adapt a vector of step sizes

**Expected SARSA with function approximation**

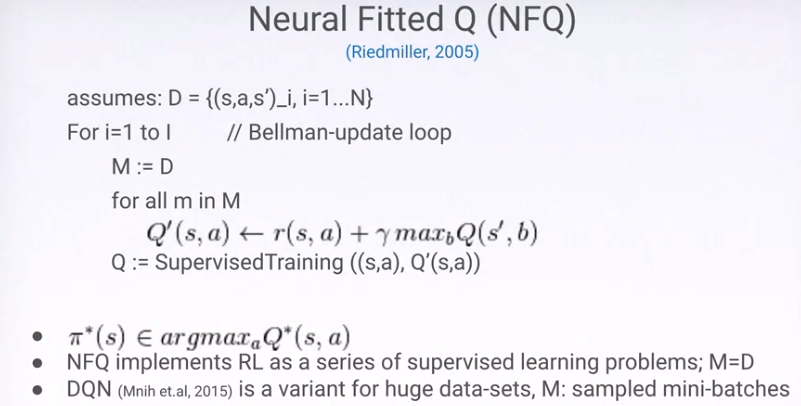
****

****

****

**Dyna and Q-learning**

**Collect and Infer framerwork – Martin Riedmiller**

****