# Generalization

Prepared by
P.Subhashree

- we assumed that the Q(s, a) values (or V (s) if we are estimating values of states) are stored in a lookup table, and the algorithms we considered earlier are called tabular algorithms. There are several problems with this approach:

(1) when the number of states and the number of actions is large, the size of the table may become quite large;

(2) states and actions may be continuous, for example, turning the steering wheel by a certain angle, and using a table, they should be discretized which may cause an error; and

(3) when the search space is large, too many episodes may be needed to fill in all the entries of the table with acceptable accuracy.

- Instead of storing the Q values as they are, we can consider this a regression problem. This is a supervised learning problem where we define a regressor $Q(s, a|\theta)$, taking s and an as inputs and parameterising by a vector of parameters, $\theta$, to learn Q values.

For example, this can be an artificial neural network with s and an as its inputs, one output, and $\theta$ its connection weights.

- A good function approximator has the usual advantages and solves the problems discussed previously. A good approximation may be achieved with a simple model without explicitly storing the training instances; it can use continuous inputs and allow generalization. If we know that similar (s, a) pairs have similar Q values, we can generalize from past cases and come up with good Q(s, a) values even if that state-action pair has never been encountered before.

- To be able to train the regressor, we need a training set. In the case of Sarsa(0), we saw before that we would like Q(st, at ) to get close to rt+1 + γQ(st+1, at+1). So, we can form a set of training samples where the input is the state-action pair (st, at ) and the required output is rt+1 + γQ(st+1, at+1). We can write the squared error as:

$$E^t(\boldsymbol{\theta}) = [r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]^2$$

- Training sets can similarly be defined for Q(0) and TD(0), where in the latter case we learn V (s), and the required output is rt+1 − γV (st+1). Once such a set is ready, we can use any supervised learning algorithm for learning the training set.
- If we are using a gradient-descent method, as in training neural networks, the parameter vector is updated as

$$\Delta\boldsymbol{\theta} = \eta[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]\nabla_{\boldsymbol{\theta}_t} Q(s_t, a_t)$$

- This is a one-step update. In the case of Sarsa(λ), the eligibility trace is also taken into account:

$$\Delta\boldsymbol{\theta} = \eta\delta_t \boldsymbol{e}_t$$

where the temporal difference error is $\delta_t = r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)$

and the vector of eligibilities of parameters is updated as $\boldsymbol{e}_t = \gamma\lambda \boldsymbol{e}_{t-1} + \nabla_{\boldsymbol{\theta}_t} Q(s_t, a_t)$

- with e0 all zeros. In the case of a tabular algorithm, the eligibilities are stored for the state-action pairs because they are the parameters (stored as a table). In the case of an estimator, eligibility is associated with the parameters of the estimator. We also note that this is very similar to the momentum method for stabilizing backpropagation.
- The difference is that in the case of momentum previous weight changes are remembered, whereas here previous gradient vectors are remembered. Depending on the model used for Q(st, at ), for example, a neural network.
- In theory, any regression method can be used to train the Q function, but the particular task has several requirements. First, it should allow generalization; we need to guarantee that similar states and actions have similar Q values.
- This also requires a good coding of s and a, as in any application, to make the similarities apparent. Second, reinforcement learning updates provide instances one by one and not as a whole training set, and the learning algorithm should be able to do individual updates to learn the new instance without forgetting what has been learned before.

- For example, a multilayer perceptron using backpropagation can be trained with a single instance only if a small learning rate is used. Or, such instances may be collected to form a training set and learned altogether but this slows down learning as no learning happens while a sufficiently large sample is being collected.
- Because of these reasons, it seems a good idea to use local learners to learn the Q values.

For example,

in radial basis functions, information is localized and when a new instance is learned, only a local part of the learner is updated without possibly corrupting the information in another part. The same requirements apply if we are estimating the state values as $V(st|\theta)$.

# Partially Observable States

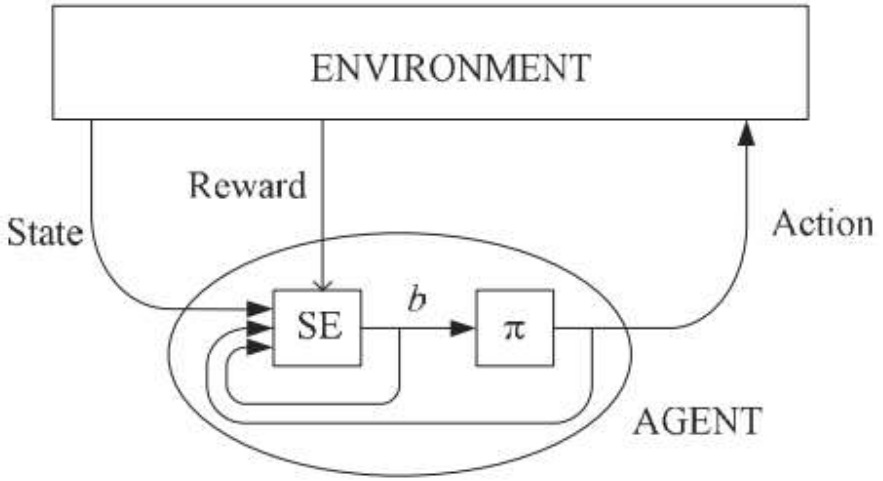
## The Setting

- In certain applications, the agent does not know the state exactly. It is equipped with sensors that return an observation, which the agent then uses to estimate the state. Let us say we have a robot that navigates in a room.
- The robot may not know its exact location in the room, or what else is there in the room. The robot may have a camera with which sensory observations are recorded. This does not tell the robot its state exactly but gives some indication as to its likely state.

  For example, the robot may only know that there is an obstacle to its right.
- The setting is like a Markov decision process, except that after taking an action at, the new state st+1 is not known, but we have an observation ot+1 that is a stochastic function of st and at p(ot+1|st, at ). This is called a partially observable MDP (POMDP).

- If ot+1 = st+1, then POMDP reduces to MDP. This is just like the distinction between observable and hidden Markov models and the solution is similar; from the observation, we need to infer the state (or rather a probability distribution for the states) and then act based on this.
- If the agent believes that it is in state s1 with probability 0.4 and in state s2 with probability 0.6, then the value of any action is 0.4 times the value of the action in s1 plus 0.6 times the value of the action in s2.
- The Markov property does not hold for observations. The next state observation does not only depend on the current action and observation. When there is limited observation, two states may appear the same but are different and if these two states require different actions, this can lead to a loss of performance, as measured by the cumulative reward.
- The agent should compress the past trajectory into a current unique estimate. These past observations can also be taken into account by taking a past window of observations as input to the policy, or one can use a recurrent neural network to maintain the state without forgetting past observations.

- At any time, the agent may calculate the most likely state and take an action accordingly. Or it may take an action to gather information and reduce uncertainty, for example, searching for a landmark, or stopping to ask for directions.
- This implies the importance of the value of information, the value of information and indeed POMDPs can be modelled as dynamic influence diagrams. The agent chooses between actions based on the amount of information they provide, the amount of reward they produce, and how they change the state of the environment.



- To keep the process Markov, the agent keeps an internal belief state bt that summarizes its experience. The agent has a state estimator that updates the belief state bt+1 based on the last action at, current observation ot+1, and its previous belief state bt . There is a policy π that generates the next action at+1 based on this belief state, as opposed to the actual state that we had in a completely observable environment.
- The belief state is a probability distribution over states of the environment given the initial belief state (before we did any actions) and the past observation-action history of the agent (without leaving out any information that could improve the agent's performance). Q learning in such a case involves the belief in state-action pair values, instead of the actual state-action pairs:

$$Q(b_t, a_t) = E[r_{t+1}] + \gamma \sum_{b_{t+1}} P(b_{t+1}|b_t, a_t) V(b_{t+1})$$

# Example: The Tiger Problem

- Let us say we are standing in front of two doors, one to our left and the other to our right, leading to two rooms. Behind one of the two doors, we do not know which, there is a crouching tiger, and behind the other, there is a treasure. If we open the door of the room where the tiger is, we get a large negative reward, and if we open the door of the treasure room, we get some positive reward.
- The hidden state, zL, is the location of the tiger. Let us say p denotes the probability that tiger is in the room to the left and therefore, the tiger is in the room to the right with a probability 1 − p:

$$p \equiv P(z_L = 1)$$

- The two actions are aL and aR, which respectively correspond to opening the left or the right door. The rewards are

| $r(A, Z)$ | Tiger left | Tiger right |
|---|---|---|
| Open left | −100 | +80 |
| Open right | +90 | −100 |

- We can calculate the expected reward for the two actions. There are no future rewards because the episode ends once we open one of the doors.

$$R(a_L) = r(a_L, z_L)P(z_L) + r(a_L, z_R)P(z_R) = -100p + 80(1-p)$$
$$R(a_R) = r(a_R, z_L)P(z_L) + r(a_R, z_R)P(z_R) = 90p - 100(1-p)$$

- Given these rewards, if p is close to 1, if we believe that there is a high chance that the tiger is on the left, the right action will be to choose the right door, and, similarly, for p close to 0, it is better to choose the left door.
- The two intersect for p around 0.5, and there the expected reward is approximately −10. The fact that the expected reward is negative when p is around 0.5 (when we have uncertainty) indicates the importance of collecting information.

- If we can add sensors to decrease uncertainty— that is, move p away from 0.5 to either close to 0 or close to 1—we can take actions with high positive rewards.
- That sensing action, aS, may have a small negative reward: $R(aS) = -1$; this may be considered as the cost of sensing or equivalent to discounting future reward by $\gamma < 1$ because we are postponing taking the real action (of opening one of the doors).