Machine Learning

10. week

- Reinforcement Learning
  - Q Learning
  - TD Learning
- Learning Vector Quantization (LVQ)
  - LVQ2

Reinforcement Learning

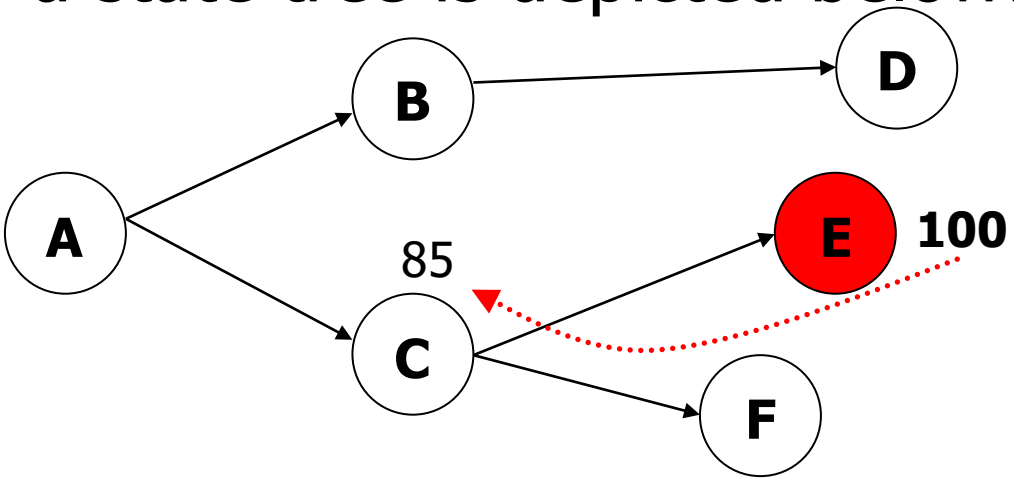
Reinforcement Learning represents, how a system with the ability to detect its environment and take decisions itself, can learn to take the right decisions in order to reach its target. This method is widely used in robotics, game programming, medical diagnosis and manufacturing automation.

Reinforcement Learning

There exists a supervisor in Reinforcement learning but unlike supervised learning it can not or does not provide enough details. Instead of this, when the system gives a decision, system is awarded for the right decisions and penalized for the wrong ones. The main purpose is to check whether the possible trials reach the target and to memorize all the right and wrong states.

Reinforcement Learning

If the decisions given are remembered as the state sequences, the successful result distributes a share from the big reward to the memorized decisive states in the sequence. An example of a state tree is depicted below:



Reinforcement Learning

Usually, there is a value (i.e. target) function (V) which determines the reward and penalty. By using behaviour policy (Π), in 't' time as state (s_t), the optimum one can be selected from the next possible states (a_t). Reinforcement learning, prefers the behaviour policy which has the maximum reward produced by the value function. Preference of the optimum behaviour policy can be formulated as:

$$\Pi = \arg\max_{a_t} \left(V(s_t, a_t) \right)$$

Q Learning

Q-learning is the most popular approach within Reinforcement learning methods. Frequently it is applied to labyrinth and search problems. In 1989 Watkins proposed this method and he used the label Q as value function, which gives its name to the method. Because of the difficulty in determining the mathematical model of the value function (Q), rewards depending on the instant situations are used.

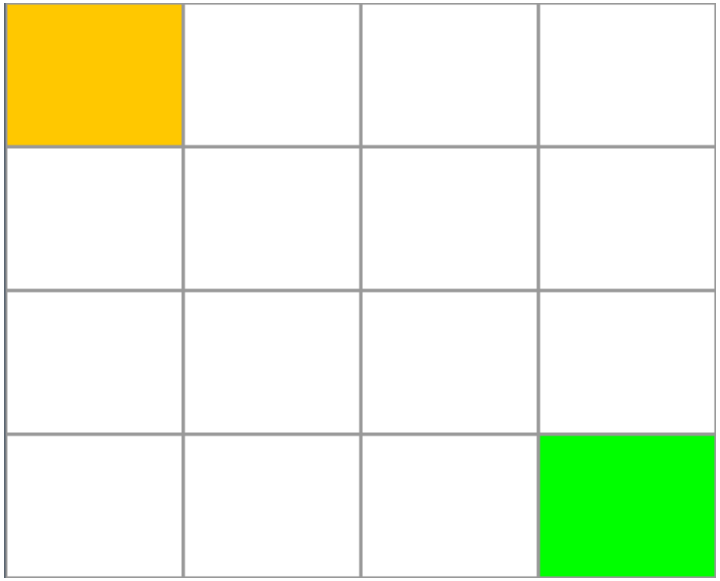
Q Learning

A reward value (r_t) can be given to the recent state. But the real value of the reward is clear when the maximum reward is reached. Actions reaching to the maximum reward benefit from it according to their distance. This discount factor (γ) is generally selected between [0 1] interval. Update of the value function can be given as:

$$Q(s_t, a_t) = r_t + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1})$$

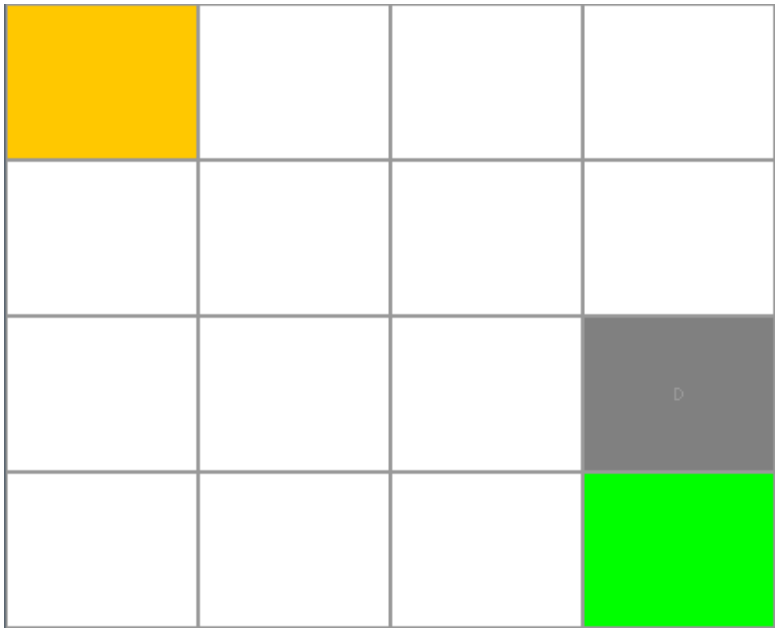
Example

As a classical problem let's see a robot searching for the exit at the right bottom of a 4x4 matrice by beginning from the left upper cell.



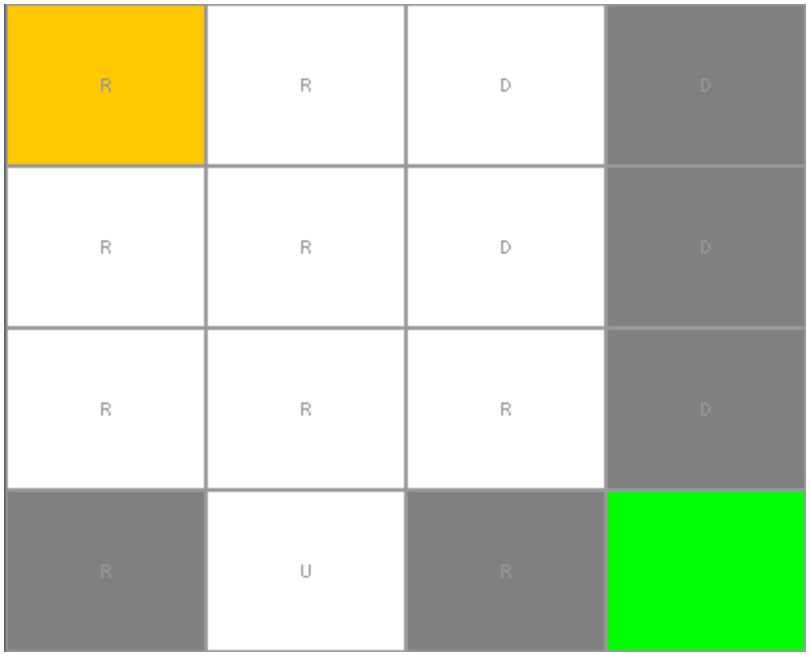
Example

Robot discovers the output by random estimations while making short-walks between the cells. It marks the cells on the way to the output.



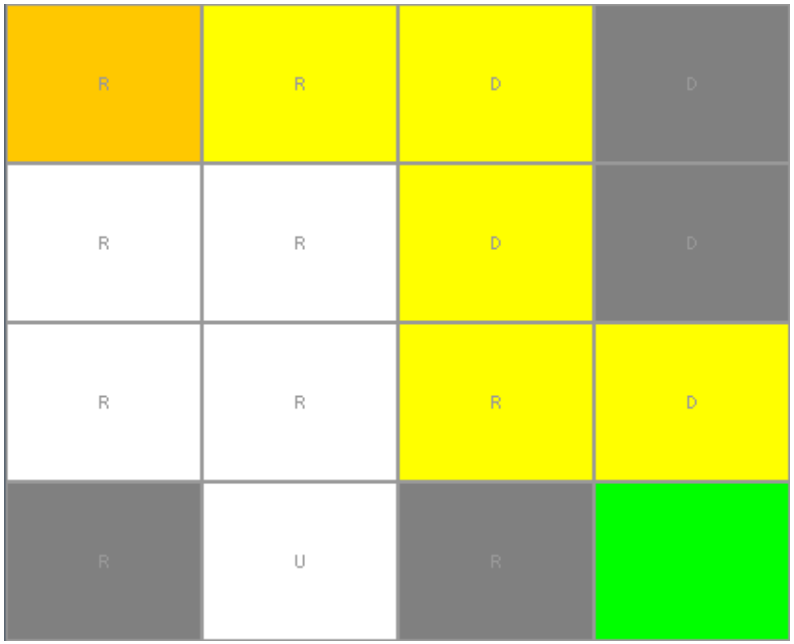
Example

Robot marks only one cell during random walks.



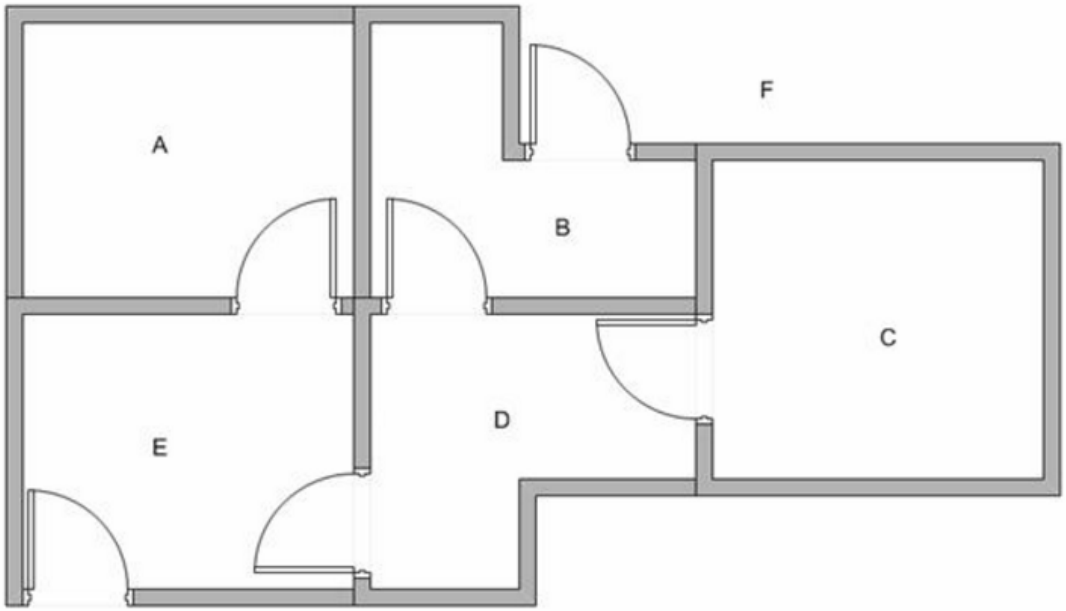
Example

At the end of the all iterations, robot finds the output as shown below:



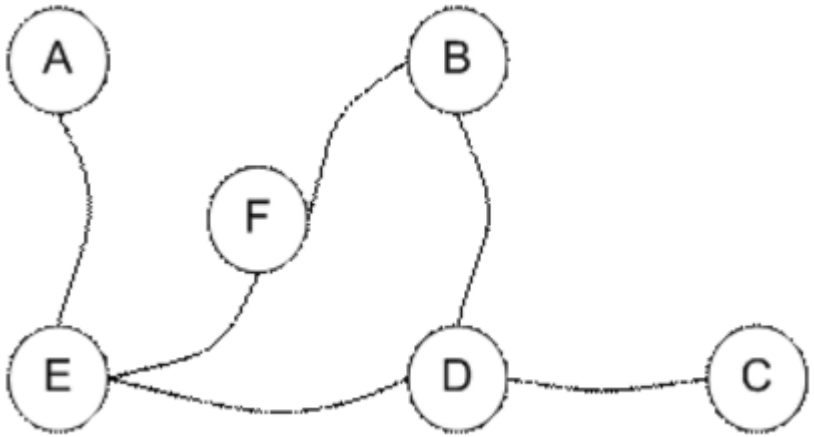
Example

Suppose we have 5 rooms in a building connected by certain doors as shown in the figure below. We give name to each room A to E. F is the exit and doors at rooms B and E can lead to room F.



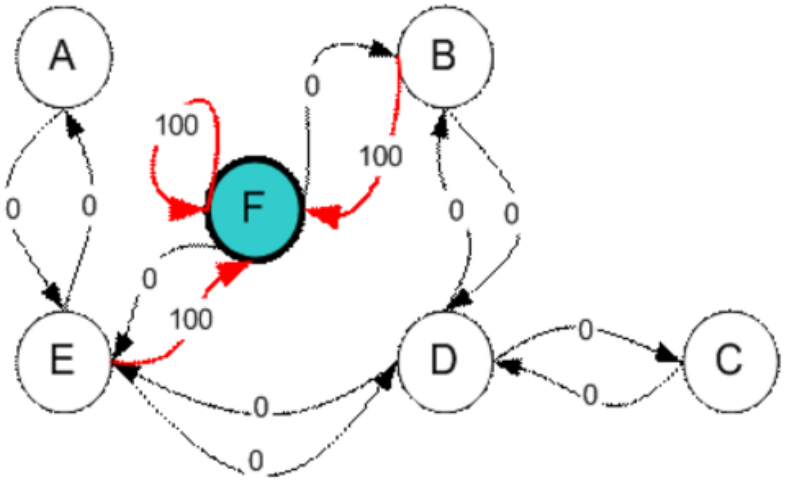
Example

We can represent the rooms by graph, each room as a vertex (or node) and each door as an edge (or link)



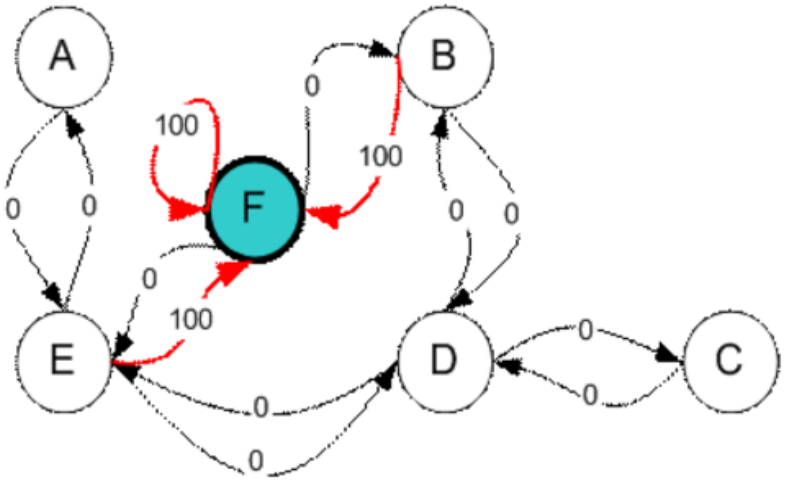
Example

If we put an agent in any room,we want the agent to go outside (node F). To set this kind of goal we give a reward value to each door (i.e. edge).The doors that lead immediately to the goal have instant reward of 100 others have 0. Each row contains an instant value.The graph becomes a state diagram as below:



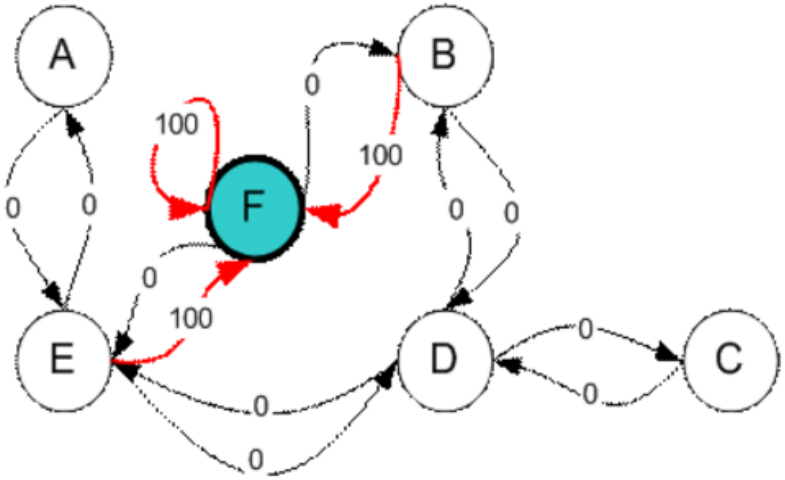
Example

We call each room (including outside the building) as a state. Agent's movement from one room to another room is called action. In the graph state is depicted using node in the state diagram, while action is represented by the arrow.



Example

How to make our agent learn from experience?Suppose the agent is in state C. From state C, the agent can go to state D, however cannot directly go to state B because there is no direct connection between B and C.If the agent is in B, it can go either to state F or D. From state A it can only go to state E and back.



Example

We can put the state diagram and the instant reward values into the following reward table, or matrix R. The minus sign in the table says that the row has no action to go to column state.

$$\mathbf{R} = \begin{array}{c} \textit{state} \backslash \textit{action} \\ A \\ B \\ C \\ D \\ E \\ F \end{array} \begin{array}{c} \begin{array}{cccccc} A & B & C & D & E & F \end{array} \\ \begin{bmatrix} - & - & - & - & 0 & - \\ - & - & - & 0 & - & 100 \\ - & - & - & 0 & - & - \\ - & 0 & 0 & - & 0 & - \\ 0 & - & - & 0 & - & 100 \\ - & 0 & - & - & 0 & 100 \end{bmatrix} \end{array}$$

Example

Now we need to put similar matrix named Q in the brain of our agent that represents the memory of the agent. The row of matrix Q represents current state the columns point to the action to go to the next state. In the beginning, we say that the agent knows nothing, thus we put Q as zero matrix.

$$\mathbf{Q} = \begin{array}{c} \\ A \\ B \\ C \\ D \\ E \\ F \end{array} \begin{array}{c} \begin{array}{cccccc} A & B & C & D & E & F \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

Example

The transition rule of this Q learning is a very simple formula:

$$\mathbf{Q}(state, action) = \mathbf{R}(state, action) + \gamma \cdot Max\left[\mathbf{Q}(next\ state, all\ actions)\right]$$

The formula above have meaning that the entry value in matrix Q is equal to correponding entry of matrix R added by a multiplication of a learning parameter γ and a maximum value of Q for all action in the next state.

Example

The pseudo code of Q learning algorithm is:

Q Learning

Given : State diagram with a goal state (represented by matrix **R**)

Find : Minimum path from any initial state to the goal state (represented by matrix **Q**)

Q Learning Algorithm goes as follow

1. Set parameter γ , and environment reward matrix **R**
2. Initialize matrix **Q** as zero matrix
3. For each episode:
    - Select random initial state
    - Do while not reach goal state
        - Select one among all possible actions for the current state
        - Using this possible action, *consider* to go to the next state
        - Get maximum Q value of this next state based on all possible actions
        - Compute $\mathbf{Q}(state, action) = \mathbf{R}(state, action) + \gamma \cdot Max\left[\mathbf{Q}(next\ state, all\ actions)\right]$
        - Set the next state as the current state

End Do

End For

Example

Let us set the value of learning paramete $\gamma = 0.8$ and initial state as room B. First we set matrix Q as a zero matrix:

$$\mathbf{Q} = \begin{array}{c} \\ A \\ B \\ C \\ D \\ E \\ F \end{array} \begin{array}{c} \begin{array}{cccccc} A & B & C & D & E & F \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

$$\mathbf{R} = \begin{array}{c} state \setminus action \\ A \\ B \\ C \\ D \\ E \\ F \end{array} \begin{array}{c} \begin{array}{cccccc} A & B & C & D & E & F \end{array} \\ \begin{bmatrix} - & - & - & - & 0 & - \\ - & - & - & 0 & - & 100 \\ - & - & - & 0 & - & - \\ - & 0 & 0 & - & 0 & - \\ 0 & - & - & 0 & - & 100 \\ - & 0 & - & - & 0 & 100 \end{bmatrix} \end{array}$$

Example

Look at the second row (state B) of matrix **R.** There are two possible actions for the current state B, that is to go to state D, or go to state F. By random selection, we select to go to F as our action.

Now we consider that suppose we are in state F. Look at the sixth row of reward matrix **R** (i.e. state F). It has 3 possible actions to go to state B, E or F.

$$\mathbf{Q}(state, action) = \mathbf{R}(state, action) + \gamma \cdot Max\left[\mathbf{Q}(next\ state, all\ actions)\right]$$

$$\mathbf{Q}(B,F) = \mathbf{R}(B,F) + 0.8 \cdot Max\left\{\mathbf{Q}(F,B), \mathbf{Q}(F,E), \mathbf{Q}(F,F)\right\} = 100 + 0.8 \cdot 0 = 100$$

Since matrix Q that is still zero, $\mathbf{Q}(F,B), \mathbf{Q}(F,E), \mathbf{Q}(F,F)$ are all zero. The result of computation $\mathbf{Q}(B,F)$ is also 100 because of the instant reward.

Example

The next state is F, now become the current state. Because F is the goal state, we finish one episode. Our agent's brain now contain updated matrix **Q** as

$$\mathbf{Q} = \begin{array}{c} \\ A \\ B \\ C \\ D \\ E \\ F \end{array} \begin{array}{c} \begin{array}{cccccc} A & B & C & D & E & F \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 100 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

For the next episode, we start with initial random state. This time for instance we have state D as our initial state.

Look at the fourth row of matrix **R;** it has 3 possible actions, that is to go to state B, C and E. By random selection, we select to go to state B as our action.

Example

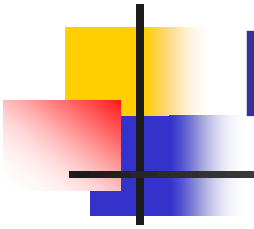
Now we imagine that we are in state B. Look at the second row of reward matrix **R** (i.e. state B). It has 2 possible actions to go to state D or state F. Then, we compute the Q value

$$\mathbf{Q}(state, action) = \mathbf{R}(state, action) + \gamma \cdot Max\left[\mathbf{Q}(next\ state, all\ actions)\right]$$

$$\mathbf{Q}(D,B) = \mathbf{R}(D,B) + 0.8 \cdot Max\left\{\mathbf{Q}(B,D), \mathbf{Q}(B,F)\right\} = 0 + 0.8 \cdot Max\left\{0, 100\right\} = 80$$

We use the updated matrix Q from the last episode. $\mathbf{Q}(B,D) = 0$ and $\mathbf{Q}(B,F) = 100$. The result of computation $\mathbf{Q}(D,B) = 80$ because of the reward is zero. The Q matrix becomes

$$\mathbf{Q} = \begin{array}{c} \\ A \\ B \\ C \\ D \\ E \\ F \end{array} \begin{array}{c} \begin{array}{cccccc} A & B & C & D & E & F \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 100 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 80 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$



Example

The next state is B, now become the current state. We repeat the inner loop in Q learning algorithm because state B is not the goal state.

There are two possible actions from the current state B, that is to go to state D, or go to state F. By lucky draw, our action selected is state F.

Now we think of state F that has 3 possible actions to go to state B, E or F. We compute the Q value using the maximum value of these possible actions.

$$\mathbf{Q}(state, action) = \mathbf{R}(state, action) + \gamma \cdot Max\left[\mathbf{Q}(next\ state, all\ actions)\right]$$

$$\mathbf{Q}(B,F) = \mathbf{R}(B,F) + 0.8 \cdot Max\{\mathbf{Q}(F,B), \mathbf{Q}(F,E), \mathbf{Q}(F,F)\}$$

$$= 100 + 0.8 \cdot Max\{0,0,0\} = 100$$

The entries of updated Q matrix contain $\mathbf{Q}(F,B), \mathbf{Q}(F,E), \mathbf{Q}(F,F)$ are all zero. The result of computation $\mathbf{Q}(B,F)$ is also 100 because of the instant reward. This result does not change the Q matrix.

Example

Because F is the goal state, we finish this episode. Our agent's brain now contain updated matrix Q as

$$\mathbf{Q} = \begin{array}{c} \\ A \\ B \\ C \\ D \\ E \\ F \end{array} \begin{array}{c} \begin{array}{cccccc} A & B & C & D & E & F \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 100 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 80 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

Example

If our agent learns more and more experience through many episodes, it will finally reach convergence values of Q matrix as

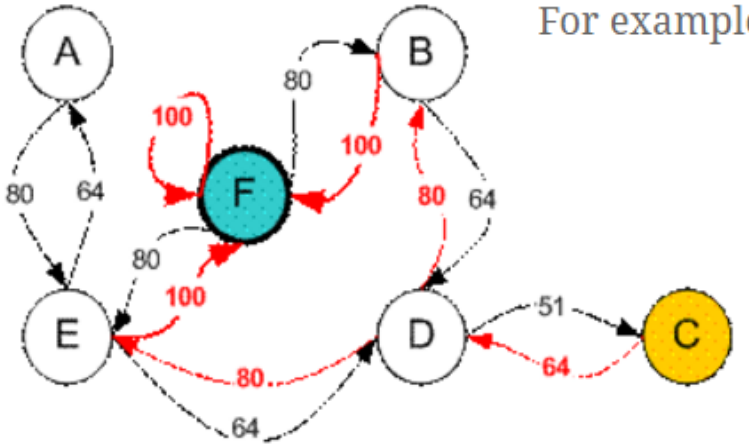
$$\mathbf{Q} = \begin{array}{c} \textit{state} \backslash \textit{action} \\ A \\ B \\ C \\ D \\ E \\ F \end{array} \begin{array}{c} \begin{array}{cccccc} A & B & C & D & E & F \end{array} \\ \begin{bmatrix} - & - & - & - & 400 & - \\ - & - & - & 320 & - & 500 \\ - & - & - & 320 & - & - \\ - & 400 & 256 & - & 400 & - \\ 320 & - & - & 320 & - & 500 \\ - & 400 & - & - & 400 & 500 \end{bmatrix} \end{array}$$

This Q matrix, then can be normalized into a percentage by dividing all valid entries with the highest number (divided by 500 in this case) becomes

$$\hat{\mathbf{Q}} = \begin{array}{c} \textit{state} \backslash \textit{action} \\ A \\ B \\ C \\ D \\ E \\ F \end{array} \begin{array}{c} \begin{array}{cccccc} A & B & C & D & E & F \end{array} \\ \begin{bmatrix} - & - & - & - & 80 & - \\ - & - & - & 64 & - & 100 \\ - & - & - & 64 & - & - \\ - & 80 & 51 & - & 80 & - \\ 64 & - & - & 64 & - & 100 \\ - & 80 & - & - & 80 & 100 \end{bmatrix} \end{array}$$

Example

Once the Q matrix reaches almost the convergence value, our agent can reach the goal in an optimum way. To trace the sequence of states, it can easily compute by finding action that makes maximum Q for this state.



For example from initial State C, it can use the Q matrix as follow:

From State C the maximum Q produces action to go to state D
From State D the maximum Q has two alternatives to go to state B or E. Suppose we choose arbitrary to go to B
From State B the maximum value produces action to go to state F

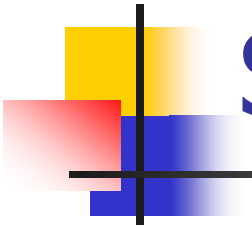
Thus the sequence is C -> D -> B -> F

TD Learning

In 1988 Sutton made an improvement in Q learning algorithm by using the delta rule and named it TD (Temporal Difference) learning. When there are more than one path heading to the reward and they intersect, the (η) parameter which is generally used for gradient descending becomes more valuable.

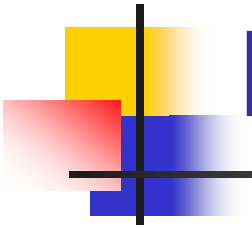
$$Q_t = Q(s_t, a_t)$$

$$Q_t = Q_t + \eta\left(r_{t+1} + \gamma \max_{a_{t+1}} Q_{t+1} - Q_t \right)$$



Sarsa: On Policy TD

TD learning can be named as a passive policy method because it selects its behaviour policy according to the next best possible action. Methods with active policy methods like Sarsa can determine next behaviour policy. Because of that it is possible to say that the problem gets one level deeper in the state tree.



Presentation Task

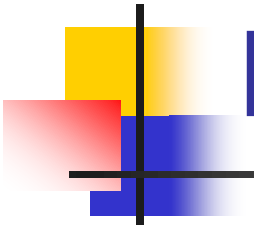
- Sarsa algorithm

Reinforced learned NN: LVQ

LVQ (Learning Vector Quantization) model, having NN architecture, is a supervised learning model inspired from Reinforcement learning

Reinforced learned NN: LVQ

Sometimes it is impossible to know about the classes of the given examples therefore it is not possible to give target values to the learned model. In the case of having the true or false value of the output which is produced by the model, classification with the Reinforcement learning is applicable.



LVQ Network

- The main objective in preparing the LVQ network is to map a NxM matrix to several vectors with M dimensions.
- Learning is processed to determine the vector set which represents the input matrice.
- In other words, the main duty of the LVQ network is to define these reference vectors by using learning.

LVQ Network Layers

LVQ is composed of three layers:

- Input Layer: External examples (Input vectors) are assigned to the network.
- Kohonen Layer: Each entity is a classified reference vector. The nearest reference vector to the input vector is defined in this layer.
- Output Layer: An output is determined for each class. Only the output for this class is activated.

Learning in LVQ Network

- Input classes in LVQ network are determined by applying the nearest 1 neighbor rule to the reference vectors.
- Generally Euclidian Distance is used as distance metric.
- Depending on the result, a reference vector is updated iteratively.

Learning in LVQ Network

- Distance between the input vector and the reference vectors is calculated.
- The reference vector with the shortest distance is determined.
- The output of this reference vector is activated at the output.
- According to the accuracy of the result, this reference vector is moved away or closer in a specified ratio.

Learning in LVQ Network

The nearest reference vector to the input vector is described as:

$$i = \arg\min_{j} \left\| x_t - v_j \right\|$$

If the output is correct, reference vector is updated:

$$v_i = v_i + \lambda\left(x_t - v_i\right)$$

If the output is not correct;

$$v_i = v_i - \lambda\left(x_t - v_i\right)$$

Learning in LVQ Network

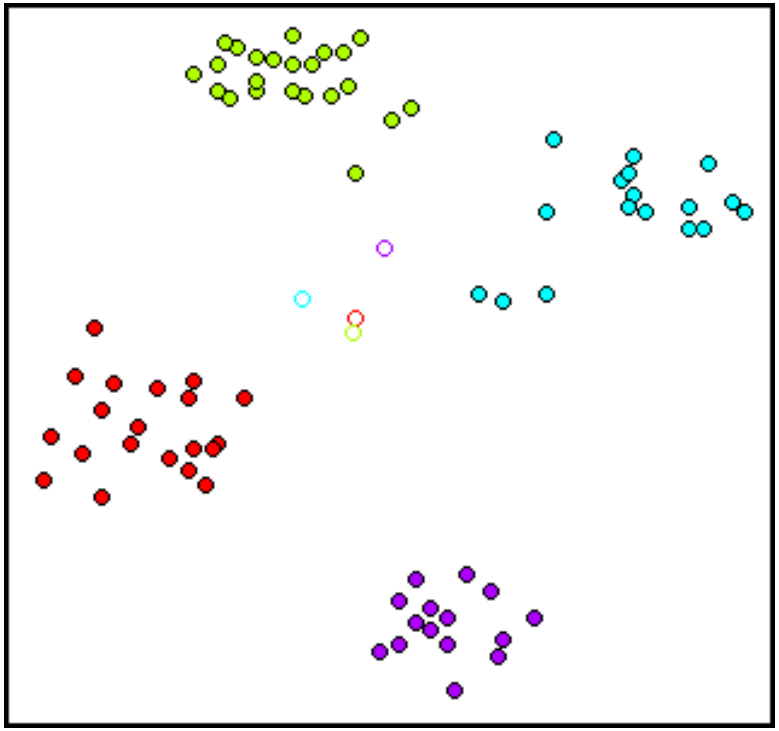
- Learning coefficient λ is generally selected between [0 1] interval.
- LVQ model having NN architecture also uses slope reduction during back propagation
- Thus λ value is decreased in each iteration

LVQ Summary

- Learning of the network is easier and faster than the basic multilayer networks. It can also be used in non-linear problems.
- If the learning coefficient does not approximate to 0 the network gets away from the correct weight values and can forget what it has learned.
- Dangerously, in some problems, the same reference vector wins ever.
- Classes of the vectors which are located between two classes or near boundaries may not be detected.

Example

Let's watch the training of the LVQ algorithm running with the popular clustering dataset Ruspini:



LVQ2 Network

- LVQ network gets difficulty in the classification of the points located between the different classes or near boundary regions. As an improvement LVQ2 is proposed.
- The nearest two vectors to the misclassified examples are detected. If the input vector is located in the window limited by these vectors and the vectors belong to different classes,it is updated.