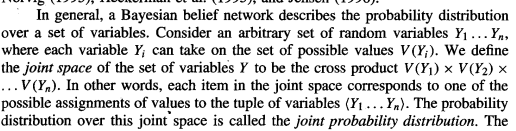
All these can be asked in essay questions

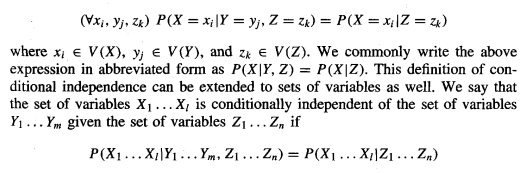
**BAYESIAN BELIEF NETWORKS (Need to explain all this )**

A Bayesian belief network describes the probability distribution governing a set of variables by specifying a set of conditional independence assumptions along with a set of conditional probabilities. In contrast to the naive Bayes classifier, which assumes that all the variables are conditionally independent given the value of the target variable, Bayesian belief networks allow stating conditional independence assumptions that apply to subsets of the variable.



Conditional Independence

Let X, Y, and Z be three discrete-valued random variables. We say that X is conditionally independent of Y given Z if the probability distribution governing X is independent of the value of Y given a value for 2; that is, if



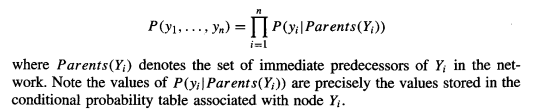
A Bayesian belief network (Bayesian network for short) represents the joint probability distribution for a set of variables. In general, a Bayesian network represents the joint probability distribution by specifying a set of conditional independence assumptions (represented by a directed acyclic graph), together with sets of local conditional probabilities.

Each variable in the joint space is represented by a node in the Bayesian network. For each variable two types of information are specified.

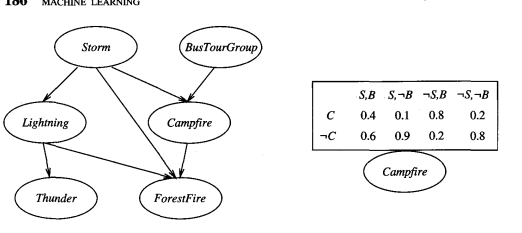
First, the network arcs represent the assertion that the variable is conditionally independent of its nondescendants in the network given its immediate predecessors in the network.

We say X is a descendant of Y if there is a directed path from Y to X.

Second, a conditional probability table is given for each variable, describing the probability distribution for that variable given the values of its immediate predecessors. The joint probability for any desired assignment of values (yl, . . . , y,) to the tuple of network variables (YI . . . Y,) can be computed by the formula



.For example, the Bayesian network in Figure 6.3 represents the joint probability distribution over the boolean variables Storm, Lightning, Thunder, ForestFire, Campjre, and BusTourGroup.



To illustrate, the Bayesian network in Figure 6.3 represents the joint probability distribution over the boolean variables Storm, Lightning, Thunder, Forest- Fire, Campfire, and BusTourGroup. C

onsider the node Campjire. The network nodes and arcs represent the assertion that CampJire is conditionally independent of its nondescendants Lightning and Thunder, given its immediate parents Storm and BusTourGroup.

This means that once we know the value of the variables Storm and BusTourGroup, the variables Lightning and Thunder provide no additional information about Campfire.

The right side of the figure shows the conditional probability table associated with the variable Campfire.

The top left entry in this table, for example, expresses the assertion that

P(Campfire = True/Storm = True, BusTourGroup = True) = 0.4

Note this table provides only the conditional probabilities of Campjire given its parent variables Storm and BusTourGroup.

The set of local conditional probability tables for all the variables, together with the set of conditional independence assumptions described by the network, describe the full joint probability distribution for the network.

The network on the left represents a set of conditional independence assumptions. In particular, each node is asserted to be conditionally independent of its nondescendants, given its immediate parents. The conditional probability table for the Campjire node is shown at the right, where Campjire is abbreviated to C, Storm abbreviated to S, and BusTourGroup abbreviated to B.

**REINFORCEMENT LEARNING**

**Questions :**

**1.Exaplain about reinforcement learning**

**2.Difference between reinforcement learning and other ML models**

**3. Explain how learning is done in reinforcement learning(need to explain briefly learning task and q learning)**

**4. Explain Q learning algorithm /Q function**

Reinforcement learning addresses the question of how an autonomous agent that senses and acts in its environment can learn to choose optimal actions to achieve its goals. This very generic problem covers tasks such as learning to control a mobile robot, learning to optimize operations in factories, and learning to play board games. Each time the agent performs an action in its environment, a trainer may provide a reward or penalty to indicate the desirability of the resulting state.

Consider building a learning robot. The robot, or agent, has a set of sensors to observe the state of its environment, and a set of actions it can perform to alter this state. For example, a mobile robot may have sensors such as a camera and sonars, and actions such as "move forward" and "turn." Its task is to learn a control strategy, or policy, for choosing actions that achieve its goals.

The goals of the agent can be defined by a reward function that assigns a numerical value-an immediate payoff-to each distinct action the agent may take from each distinct state. For example, the goal of docking to the battery charger can be captured by assigning a positive reward (e.g., +loo) to state-action transitions that immediately result in a connection to the charger and a reward of zero to every other state-action transition.

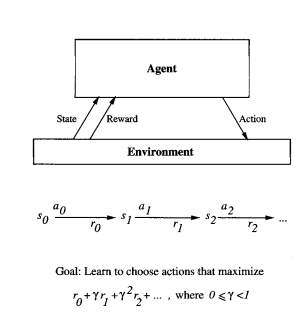


Fig: An agent interacting with its environment.

The agent exists in an environment described by some set of possible states S. It can perform any of a set of possible actions A. Each time it performs an action a, in some state st the agent receives a real-valued reward r, that indicates the immediate value of this state-action transition. This produces a sequence of states si, actions ai, and immediate rewards ri as shown in the figure. The agent's task is to learn a control policy, n : S + A, that maximizes the expected sum of these rewards, with future rewards discounted exponentially by their delay.

**Difference between Reinforcemnet Learning and Other ML Models**

Reinforcement learning problem differs from other function approximation tasks in several important respects.

**1.Delayed reward**. The task of the agent is to learn a target function n that maps from the current state s to the optimal action a = n(s). In other models, we have always assumed that when learning some target function such as n, each training example would be a pair of the form (s, n(s)).

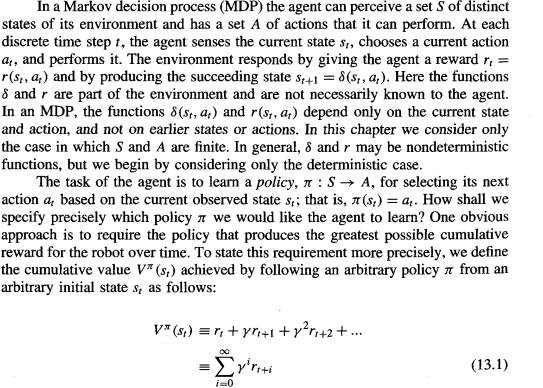
In reinforcement learning, however, training information is not available in this form. Instead, the trainer provides only a sequence of immediate reward values as the agent executes its sequence of actions. The agent, therefore, faces the problem of temporal credit assignment: determining which of the actions in its sequence are to be credited with producing the eventual rewards.

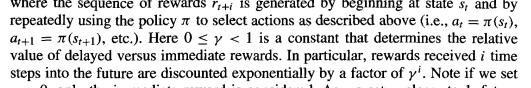
2**. Exploration**. In reinforcement learning, the agent influences the distribution of training examples by the action sequence it chooses. This raises the question of which experimentation strategy produces most effective learning. The learner faces a tradeoff in choosing whether to favor exploration of unknown states and actions (to gather new information), or exploitation of states and actions that it has already learned will yield high reward (to maximize its cumulative reward).

3. **Partially observable states**. Although it is convenient to assume that the agent's sensors can perceive the entire state of the environment at each time step, in many practical situations sensors provide only partial information. For example, a robot with a forward-pointing camera cannot see what is behind it. In such cases, it may be necessary for the agent to consider its previous observations together with its current sensor data when choosing actions, and the best policy may be one that chooses actions specifically to improve the observability of the environment.

4. **Life-long learning**. Unlike isolated function approximation tasks, robot learning often requires that the robot learn several related tasks within the same environment, using the same sensors. For example, a mobile robot may need to learn how to dock on its battery charger, how to navigate through narrow corridors, and how to pick up output from laser printers. This setting raises the possibility of using previously obtained experience or knowledge to reduce sample complexity when learning new tasks.

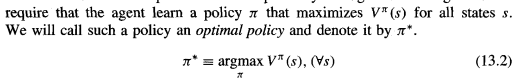
THE LEARNING TASK

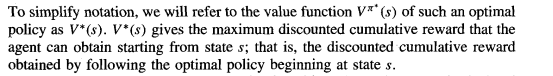




The quantity defined by Equation (13.1) is often called the discounted cumulative reward achieved by policy n from initial state s. It is reasonable to discount future rewards relative to immediate rewards.

Agents Learning Task:

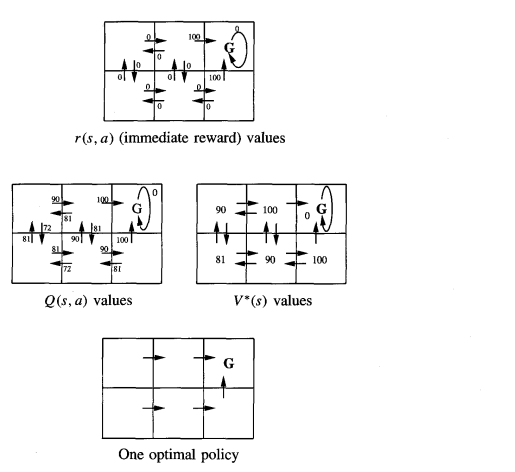




**Example:**

A simple grid-world environment is depicted in the topmost diagram of Figure 13.2. The six grid squares in this diagram represent six possible states, or locations, for the agent. Each arrow in the diagram represents a possible action the agent can take to move from one state to another. The number associated with each arrow represents the immediate reward r(s, a) the agent receives if it executes the corresponding state-action transition. Note the immediate reward in this particular environment is defined to be zero for all state-action transitions except for those leading into the state labeled G.

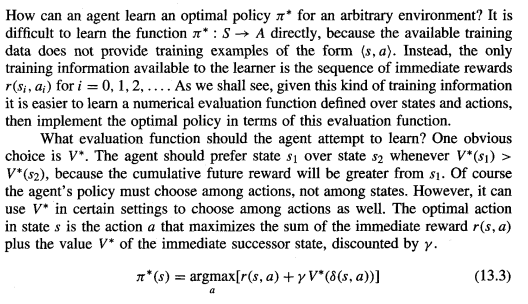
Once the states, actions, and immediate rewards are defined, and once we choose a value for the discount factor y, we can determine the optimal policy n\* and its value function V\*(s). In this case, let us choose y = 0.9. The diagram at the bottom of the figure shows one optimal policy for this setting (there are others as well).



The diagram at the right of Figure 13.2 shows the values of V\* for each state. For example, consider the bottom right state in this diagram. The value of V\* for this state is 100 because the optimal policy in this state selects the "move up" action that receives immediate reward 100. Thereafter, the agent will remain in the absorbing state and receive no further rewards. Similarly, the value of V\* for the bottom center state is 90. This is because the optimal policy will move the agent from this state to the right (generating an immediate reward of zero), then upward (generating an immediate reward of 100). Thus, the discounted future reward from the bottom center state is

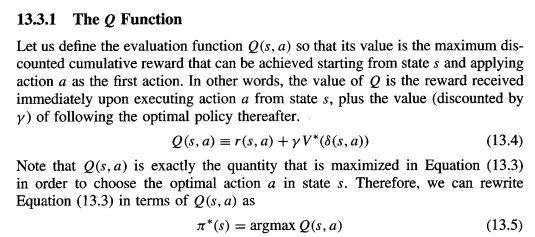


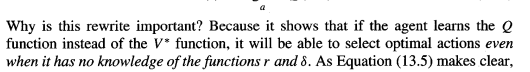
**Q LEARNING**



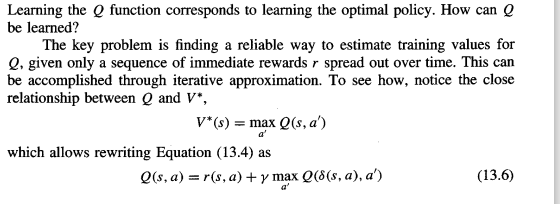


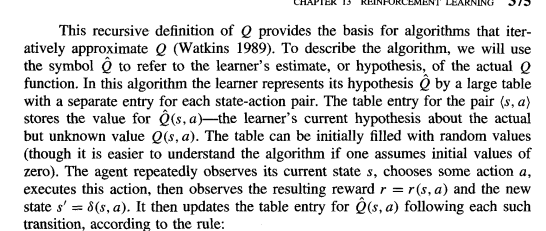




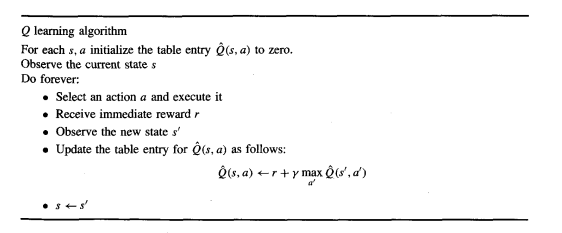


An Algorithm for Learning Q

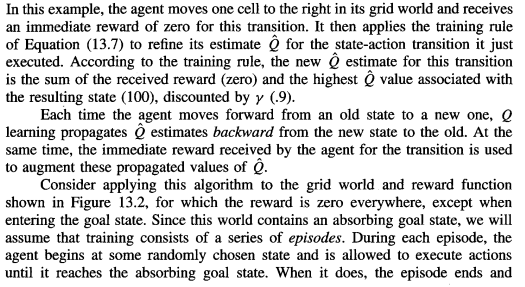
****

****

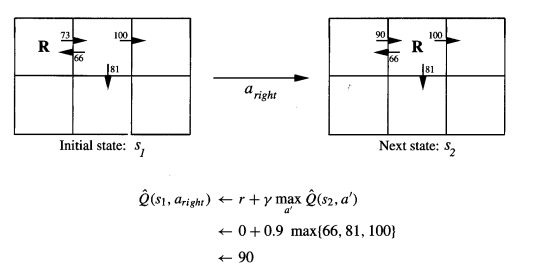
****

****

**Example:**

****

****

****

**ADVANCED TOPICS IN ARTIFICIAL NEURAL NETWORKS** (Need to explain all this including recurrent neural networks )

Gradient descent can be performed for any function E that is differentiable with respect to the parameterized hypothesis space. While the basic BAcWROPAGATION algorithm defines E in terms of the sum of squared errors of the network, other definitions have been suggested in order to incorporate other constraints into the weight-tuning rule. For each new definition of E a new weight-tuning rule for gradient descent must be derived.

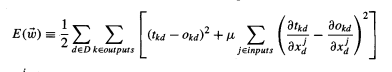
Alternative definitions of E:

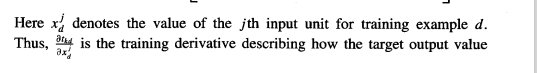
1. include a Adding a penalty term for weight magnitude. This reduces Overfitting

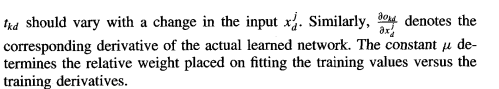


which yields a weight update rule identical to the BACKPROPAGATION rule, except that each weight is multiplied by the constant (1 - 2yq) upon each iteration

2. the error function is modified to add a term measuring the discrepancy between these training derivatives and the actual derivatives of the learned network.







**Recurrent Networks**

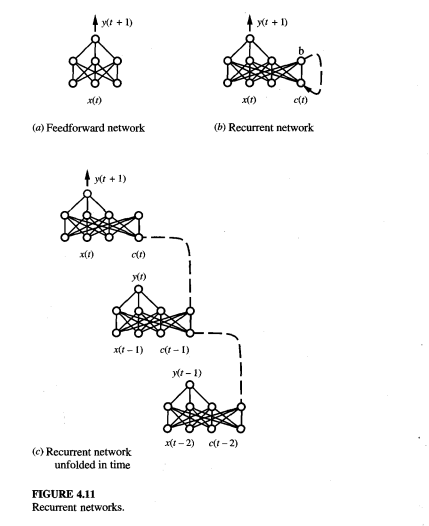
Feed forward neural network topologies are acyclic directed graphs. Recurrent networks are artificial neural networks that apply to time series data and that use outputs of network units at time t as the input to other units at time t + 1. In this way, they support a form of directed cycles in the network.

To illustrate, consider the time series prediction task of predicting the next day's stock market average y(t + 1) based on the current day's economic indicators x(t). Given a time series of such data, one obvious approach is to train a feedforward network to predict y(t + 1) as its output, based on the input values x(t).

In feed forward neural network the prediction of y(t + 1) depends only on x(t) and cannot capture possible dependencies of y (t + 1) on earlier values of x. tomorrow's stock market average y(t + 1) depends on the difference between today's economic indicator values x(t) and yesterday's values x(t - 1). This problem can be solved by making both x(t) and x(t - 1) inputs to the feedforward network.

However, if we wish the network to consider an arbitrary window of time in the past when predicting

y(t + l), then a different solution is required. The recurrent network shown in Figure 4.1 1(b) provides one such solution. Here, we have added a new unit b to the hidden layer, and new input unit c(t). The value of c(t) is defined as the value of unit b at time t – 1.

****