T-93.850 Seminar on Knowledge Engineering

Spring 2004: Reinforcement Learning

Plan

- Introduction: learning, agent, environment, reward, value function, MDP
- Q-learning
- Demonstration: maze route finding
- State generalisation, function approximation, gradient descent learning
- Demonstration: light-seeking Lego robot

Machine Learning (ML) principles

- ML usually based on constructing models that correspond to samples obtained by observing a system
- Supervised learning
  - Learn mapping from ”input” values to corresponding ”output” values
- Unsupervised learning
  - Identify relevant classes based only on ”input” values
- Reinforcement learning (RL)
  - An “agent” has to perform actions in order to collect samples from the environment

Reinforcement learning (RL)

- The learning method that resembles human and animal “higher-level” learning the most
- Neither supervised nor unsupervised learning
- Trial-and-error: an ”agent” has to take actions that sometimes result in reward (positive or negative)
- Agent goal: maximize total long-term reward
- NOT a neural network technique, but ”agent” may be implemented as neural net
- Main application areas: robotics, game playing (e.g. backgammon), linguistics

Markov Decision Process (MDP)

- (finite) MDP is a tuple $M=(S,A,T,R)$, where:
  - S is a finite set of states
  - $A = \{a_1, \ldots, a_k\}$ is a set of $k \geq 2$ actions
  - $T = [P_{sa}(\cdot) \mid s \in S, a \in A\}$ are next-state transition probabilities. $P_{sa}(s')$ is probability of transitioning to state s' when taking action a in state s
  - R specifies the reward values given when arriving to different states $s \in S$
- Most existing work assumes MDP
- States have to be **completely observable**

Value function

- Value of a state s under ”policy” π:

$$V^{\pi}(s) = E_{\pi}\left\{\sum_{k=0}^{\infty}\gamma^{k} r_{t+k+1} \middle| s_t = s\right\},$$

- where
  - s is a state representation (”inputs”)
  - r is a reward value
  - γ is a discount factor
- Gives an estimate of the sum of future reward

Action-value function

- Value of taking given action in given state
- Q-learning:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \beta\left[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)\right]$$

where

  - a_t is the action taken in state s_t, t is the ”time”
  - β is a learning factor

- Q-value is the “value” of taking action a in state s, *i.e.* an estimate of the sum of future reward

Principle of Q-learning

- Based on Temporal Difference methods, ”bootstrapping”
- If we suppose value function is continuous, then states that are ”close” should have $V(s)$ values close to each other
- Makes it possible to update value function on every step even if no reward is given

Policy

- Actions are taken following the agent's policy
- Initial policy is usually random
- A **greedy policy** always takes the action with the greatest value
- Policy should make the agent **explore** the environment so that it learns the value function
- Policy that maximizes value function is called the **optimal policy**

Exploration/exploitation tradeoff

- Policy should not become static too rapidly in order to avoid sub-optimal policies
- ε-greedy policy often used to avoid this; greedy action with probability (1- ε), random action with probability ε
- Other solutions exist

Grid-world and maze demonstration

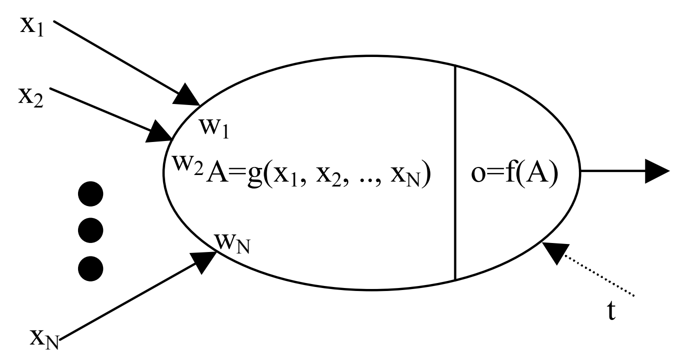
- Q-learning and ε-greedy exploration
- Also works well with stochastic state transitions
- RL is not limited to tasks with discrete states, actions and time
- BUT these are the main tasks studied in literature so far

Model representation

- In discrete-state problems: lookup-table
  - Action-value lookup-table: real-valued matrix of dimensions [actions, states]
  - The most commonly used in benchmark experiments such as grid worlds/mazes
- In tasks with continuous state or many states: artificial neural net (ANN)
  - Number of ANN outputs equals number of actions, number of inputs equals number of state variables
  - ANN output value is a Q-value estimate for corresponding action
  - Simplest ANN is linear; uses similar matrix as lookup-table
  - If non-linear regression is needed, then multi-layer ANNs can be used

Neuron

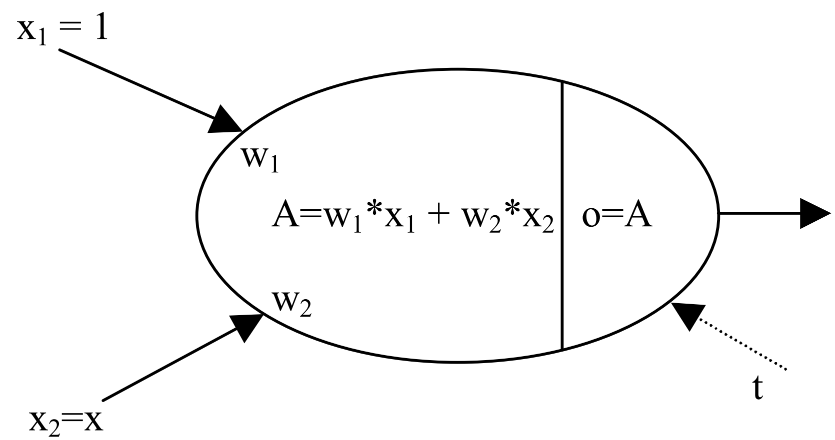
- Function `g()` calculates neuron's **activation**, A
- `g()`: usually a weighted sum of inputs $x_1 ... x_N$
- Function `f(A)` calculates neuron's output value
- `f()`: linear or non-linear (sigmoidal, Gaussian, …) function



- Learning:
  - "`t`" gives the "correct" or target output value
  - weights $w_1 ... w_N$ modifed to make difference `t - o` as small as possible

Learning with one-input, biased linear neuron

- Linear regression
- x_1 so called bias value
- $o = w_2 * x + w_1$
- $(t - o)$ iteratively reduced by modifying w_2 and w_1 in "right" direction
- Typically, w_2 and w_1 have (small) random initial values



Widrow-Hoff learning rule

- For linear neurons, i.e. f(A) = A:

$$w_i^{new} = w_i + \alpha\left(t^k - o^k\right)x_i^k$$

- , where ”`i`” is the index of the input and ”`k`” is the index of the learning example

Light-seeking robot demonstration

- 5 possible actions, 3 continuous state variables, 5x3 weights
- Only immediate reward, no delayed reward
- Task: learn sensori-motor map that makes robot turn left when the light is to the left, turn right when the light is to the right and go forward when light in front
- Seems easy, but both environment and agent are stochastic (noise) and have **hidden state**!

Combining Q-learning and ANN

- Q-learning (and other methods) handle delayed reward (temporal credit assignm.):

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \beta \left[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \right]$$

- Used target value in Widrow-Hoff learning (do not forget also multiplying by x_i^k):

$$\left[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) \right]$$

- Easy to generalize to any ANN
- Convergence of learning not certain

Conclusion

- Interesting domain because:
  - Close connection to human/animal learning
  - Mathematically challenging
  - Many potential application areas
- Most results from “simple”, simulated benchmark problems
- Real-world applications require new methods