Lecture 6

Reinforcement Learning

From Jang, Sun, Mizutani, Ch.10, Neuro-Fuzzy and Soft Computing, Prentice Hall
1. Introduction
2. Failure is the surest Path to Success
3. Temporal Difference Learning
4. Dynamic Programming
5. Adaptive Heuristic Critic
6. Q-learning
7. World Modeling
8. Other Network Configurations
Lecture Objectives

- To provide a concept of the reinforcement learning (RL) among the three kinds of learning: unsupervised-learning, supervised-learning, and reinforcement learning

- To understand several kinds of schemes in RL
1. Introduction

- Reinforcement learning (RL)
  - Machine learning
    with particular emphasis on computational aspect of learning.
  - A trial-and-error learning scheme,
    whereby a computational agent learns to perform an appropriate
    action by receiving evaluative feedback (reinforcement signal,
    performance score, grade, etc.) through interaction with the world
    that includes no explicit teacher for any correct instruction.
  - If an action is followed by an improvement in the state of affairs,
    the tendency to produce that action is strengthened or reinforced
    (rewarded).
    Otherwise, that tendency is weakened or inhibited (penalized).
1. Introduction

Four basic representative architectures:

- Policy-only
- Reinforcement comparison
- Adaptive heuristic critic (AHC, or actor-critic model)
- Q-learning

- Policy-only: The simplest RL architecture consisting solely of an adjustable policy
  - Policy: Mapping a representation of a state to an appropriate action or a probability distribution over a set of actions

- Reinforcement comparison: Effective at optimizing immediate rewards, but not at optimizing total reward in the long run. Action affects the next reward, but not the next state.

- AHC and Q-learning: To optimize long-term reward, the delayed effects of action are taken into account.
2. Failure is the surest Path to Success

- Jackpot journey
  - The objective of RL: To find an optimal policy for selecting a series of actions by means of a reward-penalty scheme
  - Suppose that journey is started at vertex A, each vertex has a signpost with a box of W/B stones (denoting “go diagonally upward (action u)”/ “downward (action d)”, and gold is placed at the terminal vertex H.
    - At vertex A, pick one stone (selection) and put it on the signpost; according to the stone’s instruction, proceed to the next vertex (action).
    - Repeat this selection-action procedure at the 2nd and 3rd vertices.
    - When the gold is found/not found, prepare a reward/penalty scheme.
    - Then, trace back to the starting vertex A; at each visited vertex, apply the reward or penalty:
      - Reward: put the placed stone back into the signpost box with an additional stone of the same color
      - Penalty: take the placed stone away from the signpost
    - Repeat the same journey many times.
2. Failure is the surest Path to Success

- Tabulate the relevant term

Path network configuration ↔ World or environment
  Traveler ↔ Agent or learner
  Vertex (intersection) ↔ State
  Picking a stone ↔ Selection an action
  Sequence of three stones ↔ Policy (or trajectory)

- Probability of action \( d \), \( P_d \) for each state:
  \[
  P_d = \frac{\text{Num}_{\text{black}}}{\text{Num}_{\text{black}} + \text{Num}_{\text{white}}}
  \]

- Probability of action \( u \), \( P_u \) for each state:
  \[
  P_u = \frac{\text{Num}_{\text{white}}}{\text{Num}_{\text{black}} + \text{Num}_{\text{white}}} = 1 - P_d
  \]
Credit assignment problems

- Through the jackpot journey, a simple reward and penalty scheme was learned.
  - This strategy is strictly success or failure driven.
  - Adjustment scheme is always applied only when the final outcome becomes known after the entire sequence of actions.
  - It ignores the intrinsic sequential structure of the problem to make adjustments at each state.
  - In playing chess, this scheme seems impractical because of the huge number of possible states chess entails.
Temporal credit assignment problem:
- In chess playing, the learner needs to make better moves with no performance indication regarding winning during the game.
- The problem of rewarding/penalizing each state (move) individually in such a long sequence toward an eventual victory or loss.

Structural credit assignment problem:
- Apportioning credit to the internal agent’s action structures
- Needs to determine which part and how much should be altered to enhance overall performance.

2. Failure is the surest Path to Success
The power of RL: By assigning immediate rewards, the agent does not have to wait until it receives feedback at the end to make adjustments.

Temporal difference methods: The states in a sequence are evaluated and adjusted according to their immediate or near-immediate successors, rather than according to their final outcomes.

- Delayed reinforcement learning
  - Deals with temporal sequence of input state vectors aimed at optimizing an evaluation function with delays between action and any resultant reinforcement.

- Immediate reinforcement learning
  - Using the most recent input-output pair alone.
  - Reinforcement comparison: Actions affect the next reward.

AHC and Q-learning:
- Take the delayed effects of action into account
Evaluation functions (EFs)

- Produce scalar values (reinforcement signals) of admissible actions at each visited state to aid in finding optimal trajectories

- Major concerns:
  - How to devise effective EFs
  - may require problem-specific knowledge
  - How to store EFs
    - can be stored as tables, symbolic rules, decision trees, CMAC, etc.
  - How to adjust EFs
    - may depend on the function forms
• Evaluation-driven agent:
  – The EFs are employed at each visited state to estimate admissible actions and how likely the actions are to lead eventually to choosing the most promising action, without concern for what the value of the final solution is.

• Result-driven agent:
  – Based on an evaluation-free algorithm
3. Temporal Difference Learning

- A class of incremental learning procedures specialized for prediction whereby credit is assigned based on the difference between temporally successive predictions.

Ordinary supervised learning \[ \cdots \quad \text{TD}(1) \leftarrow \text{TD}(\lambda) \rightarrow \text{TD}(0) \cdots \]

Classical dynamic programming

where \( \lambda \) : a discounting/recency parameter ranging from 0 to 1
3. Temporal Difference Learning

- TD formulation:
  
  - A form of supervised error-correction learning, minimizing the squared error between final outcome \( z \) and current prediction \( V_t \):
    
    \[
    E_{td} = \frac{1}{2} \{ z - V_t \}^2 = \frac{1}{2} \{ \sum_{k=t}^{m} (V_{k+1} - V_k) \}^2 \]  

  - Update rule for the parameters \( w \) of the agent’s predictor:
    
    \[
    \Delta w_t = \alpha (V_{t+1} - V_t) \sum_{k=1}^{t} \lambda^{t-k} \nabla_w V_k \]  

    where \( V_t \): the prediction value (actual output) at time \( t \),
    
    \( V_{t+1} \): the subsequent prediction (desired output),
    
    \( w \): modifiable parameters,
    
    \( \alpha \): learning rate,
3. Temporal Difference Learning

- **TD(1):** from (3.2), \( \Delta w_t = \alpha (V_{t+1} - V_t) \sum_{k=1}^{t} \nabla_w V_k \)  

  Closer to ordinary supervised learning: \( \Delta w_t = \alpha (z - V_t) \nabla_w V_t \)  

  (3.3): computed incrementally saving memory space for storing past values
  (3.4): computed not incrementally, but after completing the whole sequence of actions and storing all state-prediction pairs

- **TD(0):** from (3.2), \( \Delta w_t = \alpha (V_{t+1} - V_t) \nabla_w V_t \)

  Note “0⁰ = 1.”
3. Temporal Difference Learning

- Supervised learning poorly evaluates a new state, which is reached for the first time and then follows the trajectory (dotted line).
- TD learning reasonably evaluates the new state by adjusting the values of the observed states including $V_{p+1}(BAD)$, in a temporally successive manner.
### Jackpot journey problem

By TD(0) using the lookup-table perceptron with linearly independent state vectors $s(i), i = 1, \ldots, 6$:

- $A = s(1) = [1 \ 0 \ 0 \ 0 \ 0 \ 0]^T$
- $B = s(2) = [0 \ 1 \ 0 \ 0 \ 0 \ 0]^T$
- $F = s(6) = [0 \ 0 \ 0 \ 0 \ 0 \ 1]^T$

Apply the linear supervised learning (Widrow-Hoff) rule:

\[
V_t = w^T s_t
\]

\[
\nabla_w V_t = s_t
\]

Linear TD(0):

\[
\Delta w_t = \alpha (w^T s_{t+1} - w^T s_t) s_t
\]

where $z = 1$ for the target vertex $H$,

$z = 0$ for the other terminal vertices $G, I,$ and $J$,

$V_t$ : the expected value of each vertex = the probability of reaching gold (vertex $H$) from that vertex
3. Temporal Difference Learning

- TD(0)

Initialize $V(s)$ arbitrarily

Repeat (for each episode)
  Initialize $s$

  Repeat (for each step of episode)
    $a \leftarrow$ action given by $\pi$ for $s$
    Take action $a$, observed reward $r$, and next state $s'$
    $V(s) \leftarrow V(s) + \alpha(r + \gamma V(s') - V(s))$
    $s \leftarrow s'$

Until $s$ is terminal
Predicting cumulative outcomes

Consider a case in which each action in a sequence incurs a cost.

- At each vertex, $V_t$ is the estimate of the remaining cumulative cost rather than the total cost of the sequence.
- TD methods can be extended to deal with this case.
  - They are not confined to predicting only the final outcome of sequences, but can also be used to estimate quantities that accumulate over sequences.
3. Temporal Difference Learning

- $c_{t+1}$: the actual cost incurred between times $t$ and $t+1$
- We want $V_t$ to equal the expected value of $z_t = \sum_{k=t}^{m} c_{k+1}$.
- Prediction error = (Final outcome) – (Current prediction)
  
  \[ z_t - V_t = \sum_{k=t}^{m} (c_{k+1} - V_t) \]
  
  \[ = \sum_{k=t}^{m} (c_{k+1} + V_{k+1} - V_k) \text{ where } V_{m+1} = 0. \]

- Update rule for the following cumulative TD($\lambda$):
  
  \[ \Delta w_t = \alpha (c_{t+1} + V_{t+1} - V_t) \sum_{k=1}^{t} \lambda^{t-k} \nabla w V_k \]

- To prevent the divergence in infinite prediction problem, discounting factor can be introduced. Thus, the objective is to minimize the following discounted sum of future costs:
  
  \[ \sum_{k=0}^{\infty} \gamma^k c_{t+k+1} = c_{t+1} + \gamma c_{t+2} + \gamma^2 c_{t+3} + \cdots \]

Then, $\Delta w_t = \alpha (c_{t+1} + \gamma V_{t+1} - V_t) \sum_{k=1}^{t} \lambda^{t-k} \nabla w V_k$
3. Temporal Difference Learning

Summary

- TD can learn directly from raw experience without a model of the environment’s dynamics.
- Not require a model of the environment, of its reward and next-state probability distributions
- Updates estimates based in part on other learned estimates without waiting for a final outcome.
4. Dynamic Programming (DP)

- DP is an optimization procedure.
  - Particularly applicable to problems requiring a sequence of interrelated decisions
  - Successively approximates optimal evaluation functions by solving recurrence relations, instead of conducting searches in state space
  - Backing up state evaluations is a fundamental property of iterative procedures used to solve the recurrence relations.

- Formulation of classical DP
  - Principle of optimality:
    - *An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision*
4. Dynamic Programming

- **DP formulation:**
  - An optimal value function
  - A recurrence relation
  - Boundary conditions
  - An optimal policy function

- The optimal value function $V(s)$
  
  $V(s) = \text{The value of the minimum cost of going from a state } s \text{ to a goal } g$

- The recurrence relation
  
  - Minimum cost of a state $s$ must equal the cost of the best action $a$
    
    plus the minimal cost of the next state designated by the action $a$:
    
    $$V(s) = \min_{a \in \text{actions}} \{ \text{cost}(s, a) + V(\text{next}(s, a)) \}, \forall s \in S - G$$

    where $G$ is a set of goal states, and $S$ is the set of all states.
4. Dynamic Programming

- **Boundary condition:**
  \[ V(s) = 0, \quad \forall s \in G \]

- **Optimal policy function** \( \pi \) is \( \pi(s) = a \) such that
  \[ V(s) = \min_{a \in \text{actions}} \{\text{cost}(s, a) + V(\text{next}(s, a))\}, \forall s \in S - G \]

- For a stochastic case of this minimum-cost path problem
  - Action \( d \) with probability \( P_{d1} \); \( u \) with probability \( P_{u1} \) (\( = 1 - P_{d1} \))
  - Action \( u \) with probability \( P_{u2} \); \( d \) with probability \( P_{d2} \) (\( = 1 - P_{u2} \))
  - \( V(s) \) is the optimal expected value function, i.e. the expected value of the minimum cost of going from a state \( s \) to a goal \( g \).
  - The recurrence relation:
    \[
    V(s) = \min \left[ P_{d1} \{\text{cost}(s, d) + V(t_d)\} + P_{u1} \{\text{cost}(s, u) + V(t_u)\} \right] \\
    \left[ P_{u2} \{\text{cost}(s, u) + V(t_u)\} + P_{d2} \{\text{cost}(s, d) + V(t_d)\} \right] 
    \]
    where \( t_d = \text{next}(s, d) \), and \( t_u = \text{next}(s, u) \).
4. Dynamic Programming

- **Incremental DP**
  - In the absence of explicit probability information, we can approximate the value function
  - To improve the value function approximation $\hat{V}(s)$, we can apply incremental DP:
    \[
    \hat{V}(s) = \min_{a \in \text{actions}} \{\text{cost}(s, a) + \hat{V}(\text{next}(s, a))\}, \quad \forall s \in S - G
    \]
    \[
    \hat{V}(s) = V(s) = 0, \quad \forall s \in G: \text{boundary condition}
    \]
  - The procedure will converge to correct values if done often enough at all possible states.
4. Dynamic Programming

**Summary**

- A collection of algorithms that can be used to compute optimal policies given a perfect model of the environment as a Markov decision process
- Classical DP
  - Assumption of a perfect model and a great computational expense
- All of these methods can be viewed as attempts to achieve much the same effect as DP, only with less computation and w/o assuming a perfect model of the environment
- The curse of dimensionality
  - The number of states often grows exponentially with the number of state variables.
5. Adaptive Heuristic Critic (Actor-Critic Model)

- **Actor-critic model**
  - TD method that has a separate memory structure to explicitly represent the policy independent of the value function
  - Policy structure
    - The actor (selects actions)
  - Estimate value function
    - The critic (evaluation/prediction)

- **TD error:**
  \[
  \delta_t = r_{t+1} + \gamma V(s_{t+1}) - V(s_t)
  \]

  - \(V(s)\): value function
  - \(r\): reward
  - \(p(s,a)\): action selection probability

  TD(0): \(V(s) \leftarrow V(s) + \alpha (r + \gamma V(s') - V(s))\)
5. Adaptive Heuristic Critic (Actor-Critic Model)

- The AHC model using NN function approximators
  - Value NN
    - Approximating evaluation function mapping states to expected values
  - Action NN
    - Generating a plausible action mapping states to actions
5. Adaptive Heuristic Critic (Actor-Critic Model)

- **Algorithm**

  Observe the current state: \( s \leftarrow \text{current state} \ s_n \)

  Use the value NN to have \( V(s) \): \( e \leftarrow V(s) \)

  Select an action \( a_n \) by using the output of the action NN

  Execute the action \( a_n \)

  Observe the successive new state \( t_n \) and reinforcement \( r_n \)

  Use the value NN to compute \( V(t_n) \)

  \[
  E \leftarrow r_n + \gamma V(t_n)
  \]

  Adjust the value NN by backpropagating the error (= \( E - e \))

  Adjust the action NN according to the error

- Weight coefficients of the action NN and the value NN are adjusted to fit learned values and actions by TD methods.

- The action NN and the value NN evolve together in the attempt to find an optimal policy.
Three possible action NN architectures of the AHC-net for two-action problems

(a) Use one action NN with a single output unit
(b) Use one action NN with two output units, one unit for each action
(c) Use two action NNs with a single output unit; one NN for each action
5. Adaptive Heuristic Critic (Actor-Critic Model)

- **Exploration and action selection**
  - **Supervised learning:**
    - The agent acts according to the gradient vector provided by a teacher
  - **Reinforcement learning:**
    - No teacher supplying such directed information
    - Scalar reinforcement about the current evaluation of behavior
  - **Dilemma between exploration and exploitation**
    - Exploration: The desire to acquire more knowledge about actions’ convergences to make better selections in the future
    - Exploitation: The desire to use what is already known about the relative merits of the actions
  - To organize the desired exploratory behavior, stochastic action selection such as Boltzmann distribution should be considered:
    \[
    \text{Prob}(a_i) = \frac{\exp(V(s_i)/T)}{\sum_k \exp(V(s_k)/T)}
    \]
    where $T$ is a temperature parameter.
5. Adaptive Heuristic Critic (Actor-Critic Model)

Summary

- TD methods are employed to solve *temporal credit assignment problems*, and the backpropagation algorithm is used for *structural credit assignment problems*.

- Many of the earliest RL systems that used TD methods were actor-critic methods.

- Since then, more attention has been devoted to methods that learn action-value functions and determine a policy exclusively from the estimated values (Q-learning).
  - Advantage:
    - They require minimal computation in order to select actions (continuous-valued action)
    - They can learn an explicitly stochastic policy (competitive and non-Markov cases)
6. Q-learning

- A form of model-free reinforcement learning
  
  - Incremental version of DP:
    - Improves its evaluations of particular actions at particular states successively like incremental DP
  
  - Basic concept:
    - A simple way of solving with incomplete information Markovian action problems based on the action-value function $Q$ that maps state-action pairs to expected returns.
  
  - The aim of the agent:
    - Not merely to maximize its immediate reward in the current state, but to maximize the cumulative reward over some period of future-time
6. Q-learning

- Q-learning (here, one-step Q-learning)
  - One memory for a pair of state and action (an estimate $Q$-value of taking $a$ in $s$).
    
    *cf.* AHC: Two memories, one for evaluation function and the other one for the policy
  
  - Instead, additional complexity in determining the policy from the $Q$-values is required.
6. Q-learning

- The value of a state
  - Value of the state’s best state-action pair:
    \[ V(s) = \max_a Q(s, a) \]

- Optimal policy determined by the policy function \( \pi \):
  \[ \pi(s) = a \text{ such that } V(s) = Q(s, a) = \max_{b \in \text{actions}} Q(s, b) \]

- Advantage
  - Model-free method
    - Model: state transition probability and reinforcement function
  - Easier to implement than other RL methods
Algorithm

Initialize $Q(s, a)$
Repeat
  Start $s$
  Repeat for each step
    Choose $a$ at $s$ using policy derived from $Q$
    Take action $a$, observe $r$, $s'$
    $Q(s, a) \leftarrow Q(s, a) + \alpha \left( r + \gamma \max_{a'} Q(s', a') - Q(s, a) \right)$
    $s \leftarrow s'$
  until $s$ is terminal

where $s$: current state
$a$: action
$r$: reward
$s'$: next state
$\alpha$: learning rate, $0 < \alpha < 1$
$\gamma$: discount factor, $0 < \gamma < 1$
Ex) Grid world
Consider initial $Q = 0$, $\alpha = 0.9$, $\gamma = 0.5$, and the following rewards:

Update rule: $Q(s, a) \leftarrow Q(s, a) + \alpha (r + \gamma \max_{a'} Q(s', a') - Q(s, a))$

First trial:

Second trial:

Third trial:

First trial:

Second trial:

Third trial:
Action selection method (policy) in Q-learning

- Random
- Greedy: choose an action according to the maximum $Q$ value
- $\varepsilon$-greedy
  - Generate random probability $p$
    - If $p < \varepsilon$, then select random action,
      otherwise, select greedy action
  - Decrease probability of the threshold value ($\varepsilon$) according to the progress of Q-learning

- Stochastically implemented by a Boltzmann distribution
  - Select action with
    \[
    p(s, a_i) = \frac{e^{Q(s, a_i)/T}}{\sum_k e^{Q(s, a_k)/T}}
    \] (6.1)
  - Decrease $T$ value according to the progress of Q-learning
Algorithm: a training procedure of Q-learning NN (Q-net)

Observe the current state: \( s_n \)

Select an action \( a_n \) by stochastic procedure (6.1)

For the selected action \( a_n \), use the Q-net to compute \( U_a: U_a \leftarrow Q_{n-1}(s_n, a_n) \)

Execute the action \( a_n \)

Observe the resulting new state \( t_n \) and reinforcement \( r_n: t \leftarrow t_n \)

Use the Q-net to compute \( Q_{n-1}(t, b), b \in \text{actions} \)

\[
U \leftarrow r_n + \max_{b \in \text{actions}} Q_{n-1}(t, b)
\]

Adjust the Q-net by backpropagating the one-step error

\[
\Delta U = \begin{cases} 
U - U_a & \text{if } s = s_n \text{ and } a = a_n \\
0 & \text{otherwise}
\end{cases}
\]

\( \rightarrow \) One-step Q-learning using NN function approximation
7. World Modeling

- RL
  - Model-free (direct):
    - Learning optimal actions and values by sampling the world w/o attempting to learn a world model (e.g. transition, cost, reward models)
    - e.g. Incremental DP
  - Model-based (indirect):
    - Learning a world model by sampling the world, and then basing optimal actions and values on the learned world model
    - e.g. Classical DP
Distal supervised learning

- Can be applied to an unknown dynamic environment that intervenes between actions and desired outcomes
  - The agent views the outcomes as “distal” desired values because the agent converts reinforcement signals (called intentions) into actions, and then the environment transforms the actions into final outcomes.

- The learning agent forms a predictive internal model (forward model) by exploring the outcomes associated with particular choices of actions.
  - The forward model predicts the consequence of a given action in the context of a given state vector.

- In biological contrast, any creature with a certain sort of memory can hypothesize an action and receive a mental image of the results of that action before it is performed.
8. Other Network Configurations

- Network configurations for realizing RL
  - Divide-and-conquer methodology
    - Modular Q-network architecture
      - Decompose an entire given state space into subsets and apply different evaluation functions under different conditions.
8. Other Network Configurations

- Recurrent networks
  - The recurrence makes it possible for the networks to process sequential inputs; the RNNs may discover an intrinsic temporally successive structure of a given task.
  - The context units implicitly encode the history of the entire past.

- RL by evolutionary computation
  - Bucket brigade: assign credit based on rules that activate other rules
  - Genetic reinforcers
  - Immune modeling

cf. TD methods: assign credit based on temporal succession
Lecture Summary

- This material covered a broad range of RL techniques
  - TD method (model-free method)
  - DP is central to many techniques of RL (model-based scheme)
  - AHC model (a separate memory structure to explicitly represent the policy independent of the value function)
  - Q-learning is a model free method without recording past trials explicitly.