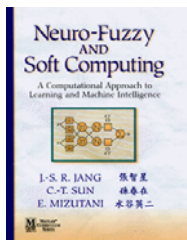
Chapter 10: Learning from Reinforcement

- Introduction (10.1)
- Failure is the surest path to success (10.2)
  - Jackpot Journey
  - Credit Assignment Problem
  - Evaluation Functions
- Temporal Difference Learning (10.3)



Jyh-Shing Roger Jang et al., *Neuro-Fuzzy and Soft Computing: A Computational Approach to Learning and Machine Intelligence*, First Edition, Prentice Hall, 1997

Introduction (10.1)

- Learning from reinforcement is a fundamental paradigm for machine learning with emphasis on computational learning load
- It is based on a trial-error learning scheme whereby a computational agent learns to perform an appropriate action by receiving a reinforcement signal (performance) through interaction with the environment
- The learner reinforces itself from lesson failures

Introduction (10.1) (cont.)

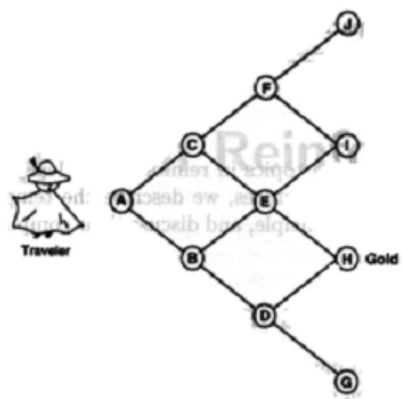
- Started & experimented in animals & particularly chimpanzees while coping with a physical environment
- It is also used by the "most intelligent" creatures on earth: the humans
- "If an action is followed by a satisfactory state of affairs, or an improvement in the state of affairs, then the tendency to produce that action is reinforced (rewarded!), Otherwise, that tendency is weakened or inhibited (penalized!)"

Introduction (10.1) (cont.)

- There are 4 basic representative architectures for reinforcement learning [Sutton 1990]
  - Policy only (probability-based actions)
  - Reinforcement comparison
  - Adaptic heuristic critic
  - Q-learning

Failure is the surest path to success (10.2)

- Jackpot journey
  - Goal: Find an optimal policy for selecting a series of actions by means of a reward-penalty scheme
  - Principle:
    - " Starting from a vertex in a graph, a traveler needs to cross the graph, vertex by vertex, in order to reach gold hidden in a terminal vertex in the graph"
    - "At each vertex, there is a signpost that has a box with some white & black stones in it. A traveler picks a stone from the signpost box & follows certain instructions; when a white stone is picked, go diagonally upward, denoted by action u. Conversely, when a black stone is chosen, go diagonally downward, denoted by action d"



The Jackpot Journey Problem

Failure is the surest path to success (10.2) (cont.)

- Jackpot journey (cont.)
  - During this travel (starting from vertex A) until one of the terminal vertices {G, H, I, J}, we have the following scheme:

    "When the gold is not found, prepare a penalty scheme. Then trace back to the starting vertex A; at each visited vertex, put the placed stone back into the signpost with an additional stone of the same color (reward), or take the placed stone away from the signpost (penalty). When the traveler returns, the next traveler will have more chances to find gold!"

Failure is the surest path to success (10.2) (cont.)

- Jackpot journey (cont.)
  - Obviously, the probability of finding an optimal policy will increase as more & more journey are undertaken

$$\mathbf{P_{down} = \frac{\#\,black\ stones}{\#\,black + \#\,white\ stones}}$$

$$\mathbf{P_{up} = 1 - P_{down}} \quad \textbf{(exclusivity \& exhaustivity)}$$

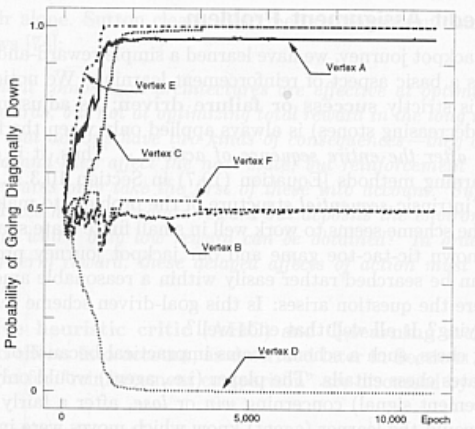


Figure 10.2. *Changing probability of action d, "go diagonally downward," at each non-terminal vertex (A–F) as trials progress in the jackpot journey problem.*

Failure is the Surest Path to Success (10.2) (cont.)

- Credit assignment problem
  - The jackpot journey is strictly success or failure driven (reward & penalty scheme!)
  - Its tuning (modification of number of stones) is performed only when the final outcome becomes known: it is supervised learning method
  - This reinforcement learning ignores the intrinsic sequential structure of the problem to make adjustments at each state

Failure is the Surest Path to Success (10.2) (cont.)

- Credit assignment problem (cont.)
  - This goal driven learning scheme is not applicable to any game playing such as "chess game"
  - In chess playing, the learner needs to make better moves with no performance indication regarding winning during the game
  - The problem of rewarding or penalizing each move (or state) individually in such a long sequence toward an eventual victory or loss is called the temporal credit assignment problem

Failure is the Surest Path to Success (10.2) (cont.)

- Credit assignment problem (cont.)
  - Apportioning credit to the internal agent's action structures is called the structural credit assignment problem
  - The structural credit assignment problem deals with the development of appropriate internal representatives
  - Temporal and structural credit are involved in any connexionist learning model such as NN

Failure is the Surest Path to Success (10.2) (cont.)

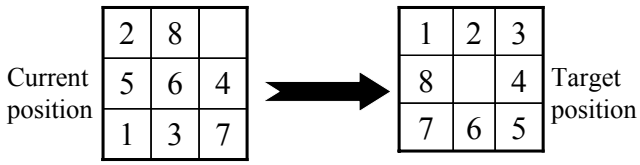
- Credit assignment problem (cont.)
  - The power of reinforcement learning lies in the fact that the learner needs not to wait until it receives feedback at the end to make adjustments (fundamental key concept!)
  - In conclusion, we need an evaluation function that gives a score to a move locally to be optimized

Failure is the Surest Path to Success (10.2) (cont.)

- Evaluation functions
  - They provide scalar values (reinforcement signals) of states to aid in finding optimal trajectories: they are emotions in the biological brain

Failure is the Surest Path to Success (10.2) (cont.)

- Evaluation functions (cont.)
  - Example: (Manhattan distance in the "eight puzzle problem")



Number of moves to reach the goal = sum of each tile's vertical & horizontal distance from its target position

tile 2 → (1,1); (1,2) → |(1-1)|+|(2-1)| = 1

tile 8 → (1,2); (2,1) → |(2-1)|+|(1-2)| = 2

This distance is used as a heuristic function in the $\mathbf{A}^*$ algorithm

Temporal Difference Learning (10.3)

- General Form: the modifiable parameter w of the agent's predictor obey the following update rule:

$$\Delta \mathbf{w}_t = \alpha\left(\mathbf{V}_{t+1} - \mathbf{V}_t\right)\sum_{k=1}^{t}\lambda^{t-k}\nabla_{\mathbf{w}}\mathbf{V}_k = \mathbf{TD}(\lambda)$$

where: Vt is the prediction value at time t

λ is a discounting parameter in [0..1]

t = 1: $\Delta w1 = \alpha\ (V2 - V1)\ \nabla wV1$

t = 2: $\Delta w2 = \alpha\ (V3 - V2)\ (\lambda\ \nabla wV1 + \nabla wV2)$

Big contribution of the most recent prediction

t = 3: $\Delta w3 = \alpha\ (V4 - V3)\ (\lambda\ 2\ \nabla wV1 + \lambda\ \nabla wV2 + \nabla wV3)$

Temporal Difference Learning (10.3) (cont.)

- More recent predictions make greater weight changes: recent stimuli should be used in combination with the current ones in order to determine actions
- TD(1): $\Delta w_t = \alpha(V_{t+1} - V_t)\sum_{k=1}^{t} \nabla_w V_k$
- TD(0): $\Delta w_t = \alpha(V_{t+1} - V_t)\nabla_w V_t$
- In TD(1) all past predictions make equal predictions to the weight alterations: all states are equally weighted whereas in TD(0) only the most recent prediction counts

Temporal Difference Learning (10.3) (cont.)

- TD(λ) can be viewed as a supervised learning procedure for the pair (current prediction Vt, desired output Vt+1) in the error term

$$E_{td} = \frac{1}{2}\{z - V_t\}^2 = \frac{1}{2}\left\{\sum_{k=t}^{m}(V_{k+1} - V_t)\right\}^2$$

where: Vm+1 = z (final outcome)

Vt = current or actual output

Since $\Delta wt \propto - \nabla w\ Etd = \nabla w\ Vt\ (z - Vt)$
which implies that: $\Delta wt = \alpha\ (z - Vt)\ \nabla w\ Vt$

Temporal Difference Learning (10.3) (cont.)

- The weight variation at time t is proportional to the difference between the final outcome & the prediction at time t
- This results shows that this scheme is similar to ordinary supervised learning: Δwt can be determined only after the whole sequence of actions has been completed (z made available!)
- Δwt cannot be computed incrementally in multiple step problems (Jackpot example)

Temporal Difference Learning (10.3) (cont.)

- However, the equation:

$$\Delta \mathbf{w}_t = \alpha(\mathbf{V}_{t+1} - \mathbf{V}_t)\sum_{k=1}^{t} \nabla_w \mathbf{V}_k$$

for:

$t = 1$: $\Delta w1 = \alpha\ (V2 - V1)\ \nabla wV1$

$t = 2$: $\Delta w2 = \alpha\ (V3 - V2)\ (\nabla wV1 + \nabla wV2)$

$t = 3$: $\Delta w3 = \alpha\ (V4 - V3)\ (\nabla wV1 + \nabla wV2 + \nabla wV3)$

provides an incremental scheme.

Temporal Difference Learning (10.3) (cont.)

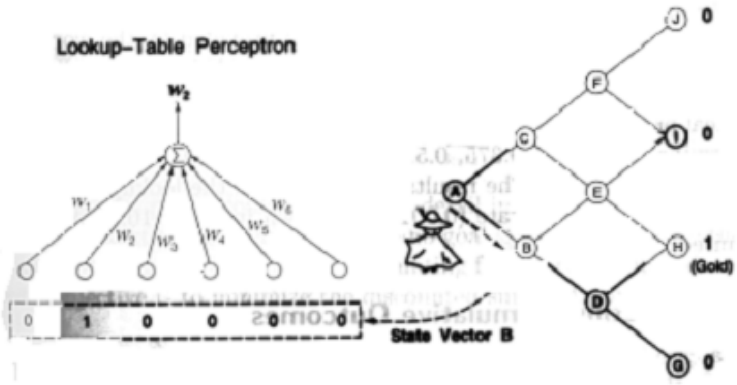
- When $\lambda = 0$, $\Delta \mathbf{w_t} = \alpha(\mathbf{V_{t+1}} - \mathbf{V_t})\nabla_{\mathbf{w}}\mathbf{V_t}$ only the most recent prediction affects the weight alteration: close to dynamic programming

Temporal Difference Learning (10.3) (cont.)

- Expected Jackpot
  - We use TD(0) to update the weights & a lookup-table perceptron to output result
    $\Rightarrow$TD perceptron
  - TD(0) provides:

$$\Delta w_t = \alpha\left(w^T s_{t+1} - w^T s_t\right)s_t$$

$$(\text{since} \quad \mathrm{V_t} = w^T * s_t \quad \text{and thus} \quad \nabla_{\mathrm{w}} V_t = s_t)$$



A lookup-table perceptron to approximate expected values in the jackpot journey problem

Table 10.1. *Predicted probabilities for six nonterminal vertices at three distinctive learning stages using a lookup-table perceptron with a small learning rate (0.001). RMSE means "root-mean-squared error."*

Epoch	Vertex A	Vertex B	Vertex C	Vertex D	Vertex E	Vertex F	RMSE
10,000	0.3109	0.4248	0.2163	0.4178	0.4718	0.0231	0.05624
20,000	0.3656	0.4922	0.2536	0.5066	0.4995	0.0019	0.00590
40,000	0.3742	0.5008	0.2496	0.5016	0.5033	0.0	0.00155
Target	0.375	0.5	0.25	0.5	0.5	0.0	0