Learning

- Machine learning
  - training
  - concept learning
  - neural networks
  - decision trees
- Reinforcement learning
  - policy search
  - Q learning
  - The pole-balancing case

Machine learning

- An agent *learns* from experience $E$, w.r.t. task set $T$, using performance measure $P$, iff:
  
  \[
  \text{Performance measure under } P \text{ on tasks in } T \text{ improves with experience } E
  \]
- Example: Driving
  - $T$: driving using vision sensors
  - $P$: average distance without error
  - $E$: images paired with steering commands
Training

- **Assumption**: Distribution of training examples matches distribution of test examples
- **Learning task**: Generalize from training examples (or past experience) to state-perspective action function

Concept learning

- **Domain**: a set, $X$, of instances of a concept
- **Target concept**: $c : X \rightarrow \{ T, F \}$
- **Training examples**: a set of pairs $(x, y)$, with $x \in X, y = c(x)$
  - Where $y = F$, example is “negative”
  - Where $y = T$, example is “positive”
- **Goal of learning**: to find $c$ by generalizing from training examples
- **Method**: Search for a hypothesis $h$ s.t. $h(x) = c(x)$ for all $x \in X$
Neural networks

- Human brain: $10^{11}$ neurons, each connected to average of $10^4$ others. Max. switching time: $10^{-3}$ sec
- Each neuron “fires” (output pulse) if summed inputs exceed a threshold
- Artificial NN ANVINN drove car at 70 mph, 1993
- Good for problems with noisy, complex sensory information (cameras, microphones)
- Most common algorithm: Backpropagation
- Slow learning, fast application
- Hard for humans to understand correspondence between problem and NN solution

Decision tree learning

- Classify instances of concepts in trees whose leaves are classifications, e.g., yes/no
- Branches test for attributes
- Robust to errors unlike other concept-learning algorithms
Learning agents

- In learning, percepts improve agent’s future success in interaction
- Components:
  - Learning element (improves policy)
  - Performance element (executes policy)
  - Critic: Applies fixed performance measure
  - Problem generator: Suggests experimental actions that will provide information to learning element

RN95, pp. 525ff

Passive ADP learning

- Adaptive dynamic programming
- Requires transition model, containing information about structure of environment
- Utility $U$ of state $i$ is computed from state’s reward $R$ and utilities of all adjacent states, weighted by transition probabilities:

$$U(i) = R(i) + \sum_j M_{ij} U(j)$$

RN95, p. 603
**Temporal difference (TD) learning**

- Uses observed transitions and differences between utilities of successive states to adjust utility estimates
- Update rule based on transition from state $i$ to $j$:
  \[ U(i) \leftarrow U(i) + \alpha (R(i) + U(j) - U(i)) \]
  where
  - $U$ is utility,
  - $R$ is reward
  - $\alpha$ is learning rate

**Model-based vs. model-free learning**

- Transition model is used in adaptive dynamic programming (ADP) learners
- Use of models is associated with knowledge-based AI
- Temporal difference (TD) learning does not use model
- Q learning is model free
Q-values

- Definition: Q-values are values $Q(a, i)$ of expected utility associated with a given action in a given state
- Utility of state:
  $U(i) = \max_a Q(a, i)$
- Q-values permit decision making without a transition model
- Q-values are directly learnable from reward percepts

TDL-based Q-learning agent

Agent attributes:
- $Q =$ table of (action, state) values
- $N =$ number of previous (action, state) visits
- $R[s] =$ reward for state $s$
- $a =$ previous action
- $s =$ previous state $s' =$ current state
- $\alpha =$ learning rate $f =$ exploratory function

Algorithm:
- $N[a, s] \leftarrow N[a, s] + 1$
- $Q[a, s] \leftarrow Q[a, s] + \alpha (R[s] + \max_a Q[a', s'] - Q[a, s])$
- $s \leftarrow s'$
- return arg $\max_{a'} f(Q[a', s], N[a', s'])$  

RN95, p. 614
Explicit vs. implicit representation

- Learned function may be represented as table (explicit)
- Policies for games with $10^{120}$ states have been represented as weights in a linear function of board features (implicit)
- Implicit representation allows generalization from states visited to those not

RN95, pp. 615-616

Value function methods vs. evolutionary methods of RL

- Evolutionary methods evaluated a policy that is fixed over many test sequences
- Value function methods update policy during execution
- Value function update benefits from information obtained during interaction

SB98, p. 13
Pole Balancing: the BOXES program

- Focused on “information-versus payoff dilemma”
- Solution to complexity: reduce complex game to simpler model, then solve subgames
- Pole balancing is “game against nature”
- Infinite continuous state set is reduced by quantization to discrete state set
- No optimal policy was known in control theory

Problem definition

- Inputs: either a state tuple or a failure signal
- State:
  - cart position (5 values)
  - angle of pole (5 values)
  - cart velocity (3 values)
  - angle rate of change (3 values)
- Actions: {left, right}, i.e., “bang-bang” control
- Interval from action to percept: 0.05 sec
Data used in learning

For each of the 225 states, a “demon” keeps score of the following, based on past inputs:

• $LL$, left life: sum of weighted durations of past trials after state was exited in a left action
• $RL$, right life
• $LU$, left usage: weighted sum of number of left actions on entry to state in past runs
• $RU$, right usage
• $Target$, desired level of attainment set by supervising demon
• $T_1 \ldots T_n$, times at which state has been entered during current run

Policy details

• Decision rule maps

\[ LL \times RL \times LU \times RU \times target \rightarrow \{\text{left, right}\} \]

• Let $N =$ # times a state has been entered this run

\[ DK = \text{weighting factor, favors recent inputs} \]
\[ K = \text{weighting factor, favors global over local inputs} \]

• Local totals are updated, global demon updates $target$

• $Value_L = \frac{(LL \times K \times target)}{(LU + K)}$; similarly for right value $Value_R$

• If $Value_L > Value_R$ then $action \leftarrow \text{left}$, else $action \leftarrow \text{right}$
Policy learning overview

- “State signal” represents percept, which contributes to state knowledge
- Policy maps from states to actions
- State value is computed using accumulated experience
- Learner adapts policy online, using this experience (*LL, LU, RL, RU, target*)

Results of [MC68]

- $DK, K$ were adjusted experimentally to 0.99, 20.0, resp.
- “Merit,” measuring success value of a trial, is computed as weighted global-live / global-usage, i.e., roughly the number of steps per trial before failure, weighted to favor later trials
- System achieved a run of over 72,000 decisions (i.e., about 1 hour of real-time control)
Discussion

• (Michie and Chambers, 1968) was early RL research aimed at solving a difficult adaptive control problem

• Insights:
  - Use of trial and error
  - Decomposition of problem
  - Separation of exploitation and exploration, rejection of greedy action
  - Use of optimistic target performance level to encourage exploration (stretch system to explore to meet high goals)

Sources


