Wednesday February 4th

ECE 485/585

Intelligent Robotics

Introduction to Learning

Outline

- Introduction
- Definitions
- - Supervised learning
- - Unsupervised learning
- -Reinforcement learning
- Conclusion

Why Learning is working?

- Computational learning theory:
  - **PAC** – a system with enough trials will be doing “probably approximately correct”, thus has learned something
- **error(h) <** λ – after n examples the system produce approximately correct outputs
- The solution considered as PAC lies in the proximity of the solution, similar to monte-carlo local search

The evaluation according to CLT Computational Learning Theory

- A bad answer/hypothesis can be described as:
  - $P(h_b \text{correct} \rightarrow N \text{examples}) \leqslant (1-\epsilon)^N$
  - Probability of a False-False evaluation
- Similalrly:
  - $P(H_b \text{contains a False-positive}) \leqslant |H_{bad}|(1-\epsilon)^N \leqslant |H|(1-\epsilon)^N$
  - Probability that Mis-hits contains correct answers

Hypothesis evaluation

- Error interpretation:
  - False-positive
    - A hypothesis answered as valid that in reality is not
  - False-negative
    - A hypothesis answered invalid that in reality is valid one
- This can be visualized as a positive or negative gradient evaluation durring learning approaches such as Neural Network

CLT (cont.)

- It can be easily derived from previous equations:
  - $|\mathrm{H}|(1-\epsilon)^{\mathrm{N}} \leqslant \delta$
  - The probability of correct answer is smaller than δ (constant)
- Sample complexity
- Training set and training perisd complexity
- etc.

Bayesian approach to Learning
A introduction..

- Calculates probability of each hypothesis- making probabilistic predictions about events
- Let set I, be the data; **i** is observed sample, then $P(h|\mathbf{i})=alphaP(\mathbf{i}|h)P(h)$ where h is the set of all hypothesis about the observed data samples **i**
- Similalrly for some unkown quantity:
  - $P(X|\mathbf{i})=\sum_i P(X|\mathbf{i},h)P(h|\mathbf{i})=\sum_i P(X|h)P(h|\mathbf{i})$
  - Keywords: hypothesis prior: P(h)

    likelihood: $P(\mathbf{i}|h)$

Bayesian (cont).

- For IID – independently and identically distributed observations it simplifies to:
  - $P(\mathbf{i}|h)=\prod_j P(i_j|h)$
  - For a set h of hypothesis about the observation distribution
- Maximum a posteriori (MAP) – basing predictions on the most probable hypothesis so as: $P(X|\mathbf{i})\approx P(X|h_{MAP})$
- Maximum Likelihood (ML) hypothesis:
- Specific (MAP) case where the learning is to maximize: $P(\mathbf{i}|H_i)$

Bayesian (cont.)

- Parametrized “Maximum-likelihood“
  - Parameter can be for example a probability ϵ of an event, thus:

$$P(\mathbf{i}|h_\epsilon)=\prod_{j=1}^{N} P(i_j|h\epsilon)=\epsilon^p*(1-\epsilon)^l$$

  - Evaluation according to maximum likelihood and log likelihood:

$$P(\mathbf{i}|h_\epsilon)=\prod_{j=1}^{N} \log P(i_j|h_\epsilon)=c\log\epsilon+l\log(1-\epsilon)$$

This is valid for discrete models, most useful in science and engineering

Bayesian (Cont.)

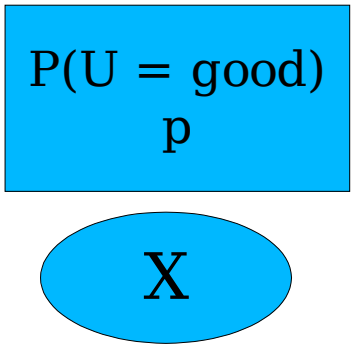
- Naïve Bayesian:
  - Assuming attributes are independent
  - Example (we want to predict L):
  - $p = P(L = \text{true}), p_{i1} = P(X_i = \text{true} | L = \text{true}), p_{i2} = P(X_i = \text{true} | L = \text{false})$
  - And the probability for each variable (model) L is given after training:

$$P(L | x_1, \ldots x_n) = \alpha P(C) \prod_i P(x_i | C)$$

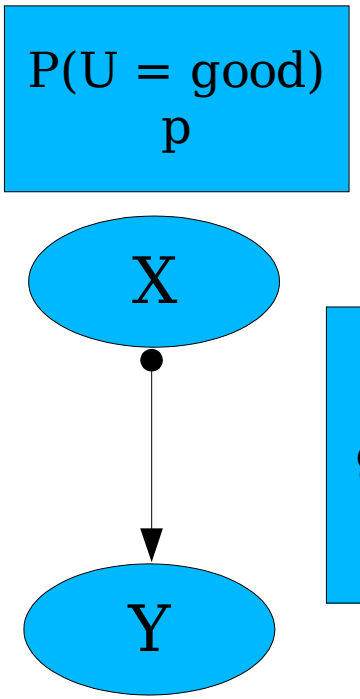
  - Models these approaches are used in: Bayesian networks

BAYESIAN NETWORKS

- Each node contains prior distribution and thus posterior probability of the given node can be computed



Example of a node with independent probability from other elements



Example of a node with conditional probability with relation to P(U = good)

F	P(V = round|U)
good	p_1
bad	p_2

Reinforcement Learning (RL)

- Agent
- Reactive Agent, autonomous agent, adaptive agent, etc.
- Agent does not know anything
    - exploration/exploitation
- Agent is a interactive entity that reacts according to its inputs, its internal state and its policy
- A agent is not a simple software but can be compared to a Demon software/code

Passive RL

- policy fixed (deterministic output)
  - learn the utility function
- Utility in this case:

$$U_\pi(s) = E\left[\sum_{t=0}^{\infty} \gamma^t R(S_t) | \pi, s_0 = s\right]$$

- Bellman equation:

$$U_\pi(s) = R(s) + \gamma \sum_{s'} T(s, \pi(s), s') U_\pi(s')$$

The approach is: out expectation must match the reality, in such a case the utility function represents the agent environment and interaction mapping/memorizing

The Bellman equation is unique because shows that my optimal policy is the result of current state/action/decision and the best expectation of future reward at the infinite horizon; t = ∞

Reinforcement learning

- DP, ADP, HADP, etc.
- TD-learning

$$U_{\pi}(4) = -0.12 + U_{\pi}(2)$$

$$U_{\pi}(s) \leftarrow U_{\pi}(s) + \alpha(R(s) + \gamma U_{\pi}(s') - U_{\pi}(s))$$

- The TD (Temporal difference) learning approach is based on the difference between the expectation of two consecutive steps and the expectation made about the step. Again the solution is to maximize locally the reward while optimizing it globally

Reinforcement learning

- DP, ADP, HADP, Adaptive critics approach:
  - Methods used for controll system because they can learn online thus suitable for industrial control
  - Often combined with stability and control theory in order to assure the security of the whole process
  - These methods are using a model of the future to predict the best current step response

Active reinforcement learning

- Passive RL: fixed policy
- Active RL: the policy changes and represents or is related to the agent internal state. The obtained behavior is much more various and able to adapt to complex phenomena

$$U(s)=R(s)+\gamma\sum_{s'}T(s,a,s')U(s')$$

- Forcing exploration:

$$U(s)=R(s)+\gamma\max_{a}f(\sum_{s'}T(s,a,s')U^{o}(s'),N(a,s))$$

- The problem in RL is the switching of policy that can be approximated by exploration and exploitation ratio. When should a RL agent do some other task among a set T of all possible tasks? Should it finnish all tasks before starting a new one?
- This is partially explored by the RL approach, Game Theory and more detailed in approaches such as Indexes from Gittins.

Q-learning

- Learning action value/policy
  - problem: lookup tables
- Can be used on most of models.

$$U(s)=\max_{a} Q(a,s)$$

- We define the utility as the quality of a certain action a on the current state s. The learning results in a lookup table where we are maximizing the value of the Utility function

$$Q(a,s)=R(s)+\gamma\sum_{s'} T(s,a,s')\max_{a'} Q(a',s')$$

$$Q(a,s)\leftarrow Q(a,s)+\alpha(R(s)+\gamma\max_{a'} Q(a',s')-Q(a',s'))$$

- The result is that the utility function is fixed by its computational nature but can be modified by assigning weight to each transition such as: $s'\leftarrow s_a$

Generalities

- Utility function example

$$U_{\phi}(x, y) = \phi_0 + \phi_1 x + \phi_2 y$$

- Utility function in the case where it is completely known
- Problem: in general the function (utility) is not known at all.
- Thus the utility function is well simulated by any learning thechnique that can be updated according to the reward or policy success.
- In general the system can be constructed as a reward predictor hwere the actions are directly implied from such a prediction. Thus by predicting the reward for each step using a Neural Network, we are in the case of supervised learning if it is assumed that the reward is obtained immediately after the action. An the modification of the network weights is made by Widrow – Hoff rule for example