Feb 16th, 2017

How does an RL agent select actions?

① purely random
↳ **con**: can't exploit.

② greedy (take the best all the time)
↳ **con**: doesn't explore

③ ε-greedy ($\varepsilon \in [0,1]$ but small)
↳ take the best action with probability $(1-\varepsilon)$ and a random action with probability ε.

con: Taking the best frequently at the beginning is not correct.

④ Softmax action selection

$$p(a) = \frac{e^{Q(s,a)/\tau}}{\sum_b e^{Q(s,b)/\tau}}$$

($p(a)$: action; $Q(s,a)$: estimated value of a in s)

(a = specific action; b = running variable 1 to # of actions)

How should an RL agent store the rewards?

The average of the first k rewards $Q_k = \frac{r_1 + r_2 \cdots r_k}{k}$

requires a lot of memory!!

Can we do this incrementally?
↳ "running sum"

$$Q_{k+1} = Q_k + \frac{1}{k+1}(r_{k+1} - Q_k)$$

New estimate = old estimate + stepsize (Target − old estimate)

How can an RL agent follow the trajectory?

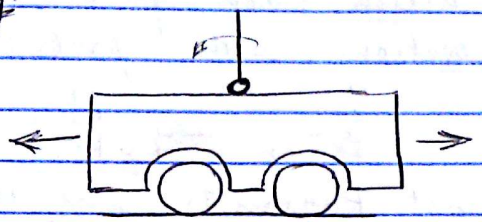
Q_k to be a simple average then this may work for a stationary problem. But for non stationary problems, it's better to use: $Q_{k+1} = Q_k + \alpha[r_{k+1} - Q_k]$

probability that 'a' is the best action when you are in state S

What should our RL agent learn?
↳ RL has to learn a policy π
$\pi(S, a)$ = probability that $a_i = a$ when $S_t = S$
but π changes as a result of more experience.

How can an RL agent function? episodic or never ending?

Eg.



task: avoid failure beyond a certain angle.

We can understand this as an episodic problem.

Each episode ends with a failure.
reward = +1 for each step before failure.
return = # of steps before failure.

Continuing Task: with discounted return.
reward = −1 upon failure
0 otherwise.
return = $-\gamma^k$ for k steps before failure.

In either case, return will be maximized by avoiding failure for as long as possible.

RL Implementation: Tabular
↳ Q Learning
↳ One step Q-Learning

Q Learning
1. Initialize $Q(S, a)$ randomly
2. Repeat (for each episode)
   2a) initialize S
   2b) Repeat (for each step of the episode)

Reinforcement learning is with you in the field. Neural Networks are not so.

(Chose an action a from set of states S using the policy π from the estimated matrix Q.)

2b1) Chose a from S using π from Q

2b2) Take action a. observe ~~r~~ r, S'

2b3) Update the Q values.

$$Q(s,a) \leftarrow Q(s,a) + \alpha\left[r + \gamma \max_{a'} Q(s',a') - Q(s,a)\right]$$

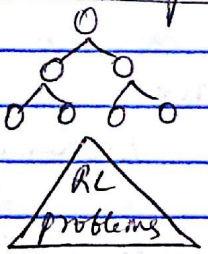
$(n+1)$ $\quad$ n $\quad$ $(n+1)$ $\quad$ (n) $\quad$ target $\quad$ $= n$

$$Q^{(n+1)}(s,a) \leftarrow Q^{(n)}(s,a) + \alpha\left[r + \gamma \max_{a'} Q^{(n)}(s',a') - Q^{(n)}(s,a)\right]$$

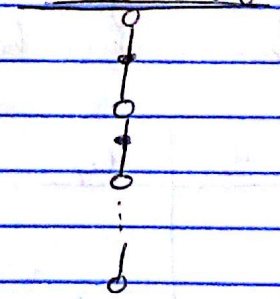
2b4) $S \leftarrow S'$

2c) Until S' is terminal.

Dynamic programming

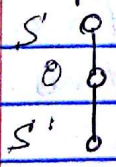


Monte Carlo



used for subsets of entire problem. no compression no dimensionality reduction.

Temporal differencing



class of algo works with a compact understanding of what the system is about

$TD(\lambda)$ is a class of temporal differencing RL methods

↳ averaging all n-steps backups

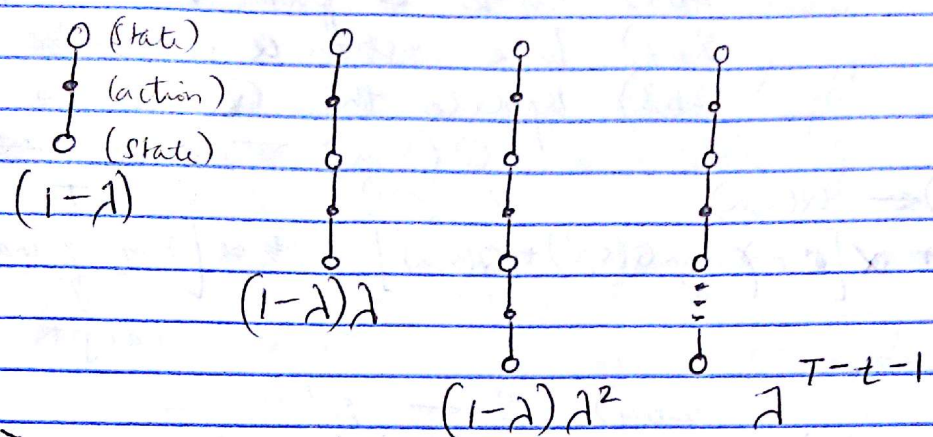
↳↳ weight by λ^{n-1} $\quad \lambda \in [0,1]$

↳↳ λ-return.

Backup using λ-return:

$$\Delta V_t(S_t) = \alpha\left[R_t^{\lambda} - V_t(S_t)\right.$$

TD(λ) λ-return:



TD($\lambda=0$) ↓ Q-learning.

* TD methods don't require a model, but experience (QLearning is a special case of TD)
QLearning is the most simplest RL algorithm (very short sighted.)
if you want far sighted, use some $\lambda > 0$
* TD methods can be fully incremental.

Q	a_1	a_2	a_3	a_4
s_1				
s_2				
→ s_3	2.1	1.3	0.9	3.7
s_4				

the table is initialized with random numbers.
→ choose this if you're greedy
every cell $q(s,a)$ has to be visited multiple times.

$\varepsilon \to 0$
$\varepsilon = 0.00001$ (never 0)
⇒ always learn. never finish learning

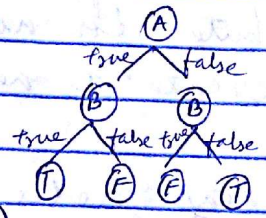
Compare: (Som v/s kmeans, auto encoders v/s rnn)

Midterm:
① Tutorial Questions: 25%
② Lecture Questions: 75%
2a) Quiz like ≈ 20% of 75%
2b) general questions: explain / describe ≈ 20%
2c) Philosophical questions: 20% (Eg: Turing v/s Chinese)
2d) Comparitive questions: 20% (this v/s that)

2e) Math questions ≈ 20% of 75%.
(entirely about Back propagation & Gradient descent)
(derivative of sigmoid function ... etc.)

Feb 28th, 2017 Decision Trees (DT)

A, B attributes



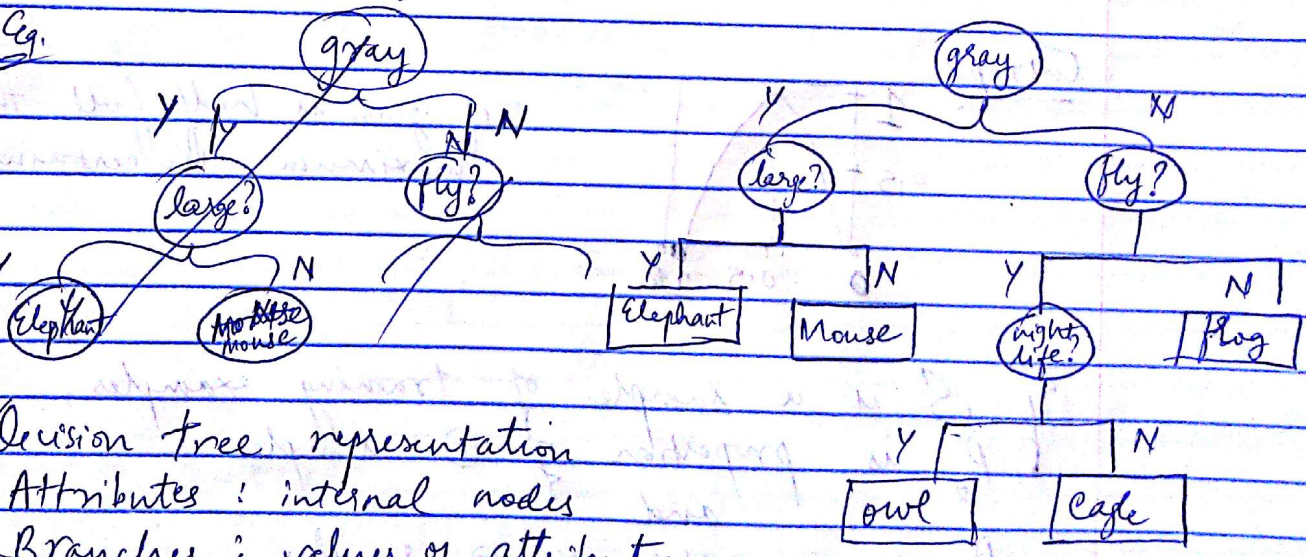
leaves (decisions)

$(A \,\&\, B) \mid (\neg A \,\&\, \neg B)$

Can we use trees (ADT) for learning to recognize scenes/objects?

"Attributes" of trees (not features)
↳ more commonly used with "trees"

Eg.



Decision tree representation
- Attributes : internal nodes
- Branches : values of attributes
- leaves : classification

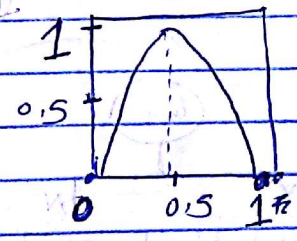
When to use DTs:
↳ instances are describable by (attribute, value) pair.
↳ Target function is discrete.
↳ Disjunctive hypothesis may be necessary
↳ noisy data.

Eg: for applications
① Credit risk analysis
② Medical diagnosis (computer aided assistance to clinical experts)
③ Modelling scheduling references.

⇒ General scheme for building a Decision Tree
① A ← the best decision attribute for the next node.
② Assign A as decision attribute for node.
③ For each value of A, create a new descendant of node.
④ Sort training examples to leaf nodes
⑤ If training examples perfectly classify, then stop, otherwise iterate over new leaf nodes.
Repeat ① to ⑤.

What does it mean for an attribute to be the best?

Entropy.



if glass is half full, there is maximum uncertainty.

If S is a sample of training examples,
$P_{\oplus}$ is proportion of $\oplus$ samples
and
$P_{\ominus}$ is proportion of $\ominus$ samples.

Entropy H measures the impurity of S.

$$H(S) = -P_{\oplus} \log_2 P_{\oplus} - P_{\ominus} \log_2 P_{\ominus}$$

The expected # of bits needed to encode class ($\oplus$ or $\ominus$) of randomly drawn member of S (shortest-length condition)

Information Theory

optimal length code assigns $-\log_2 p$ bits to a message having the probability p.

Entropy → how bad an attribute is
? → how good an attribute is.
↳ i.e. how much would I gain from adding the attribute A to my tree?

Gain (S, A) = expected reduction in entropy due to sorting on A.
(Sample) (attribute)

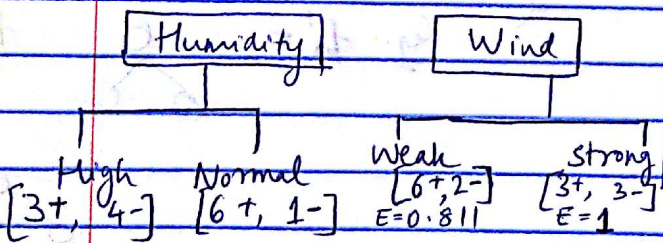
$$\text{Gain}(S, A) = H(S) - \sum_{v \in \text{Values}(A)} \frac{|S_v|}{|S|} H(S_v)$$

Refer to Eg. on LEARN.

$\|\ldots\|$ = cardinality of set, not the absolute value

For playing Tennis problem, which attribute is the best?

Sample: 9 positives (yes, play tennis)
5 negatives (No)
$S = [9+, 5-]$
$E = 0.940$



Gain $(S, \text{"Humidity"})$ #
$= 0.940 - \left(\frac{7}{14}\right) 0.985 - \left(\frac{7}{14}\right) 0.592$

$= 0.151$ → Information gained if we add "humidity" to tree.

Gain $(S, \text{"Wind"})$
$= 0.940 - \left(\frac{8}{14}\right) 0.811 - \left(\frac{6}{14}\right) 1$
$= 0.048$

∴ Humidity has most discrimination power. Keep it @ the top of decision tree

Main Question: Small or Large tree?

↳ occam's Razor

A short hypothesis that fits the data is unlikely to be coincidence.

∴ we prepare small trees.

Overfitting

Hypothesis ≡ Solution

Training data : $error_{train}(h)$

Distribution of data : $error_{\mathcal{D}}(h)$

Hypothesis $h \in H$ overfits the training data if there is $h' \in H$ such that

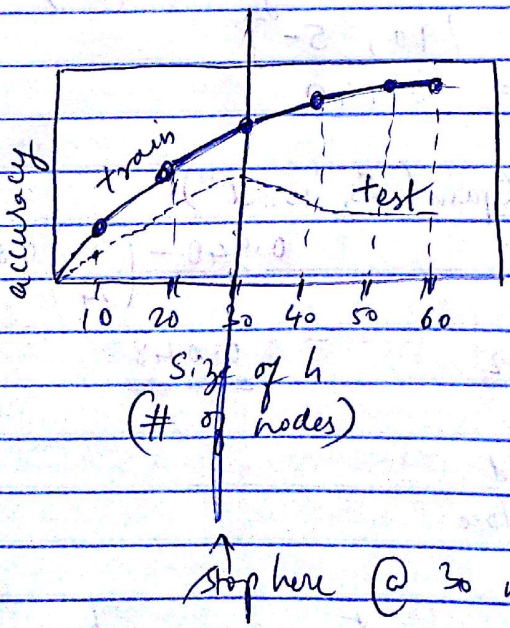
$error_{train}(h) <= error_{train}(h')$

but $e_{\mathcal{D}}(h) > error_{\mathcal{D}}(h')$.

for entire data $\mathcal{D}$

(the one that overfits is bigger one)

e.g. h' = h =



Stop here @ 30 nodes.