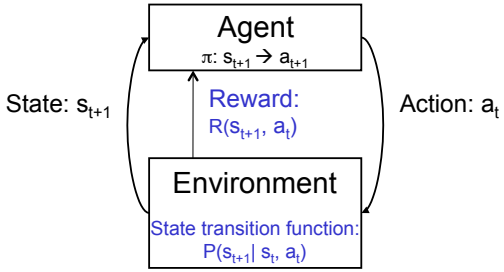
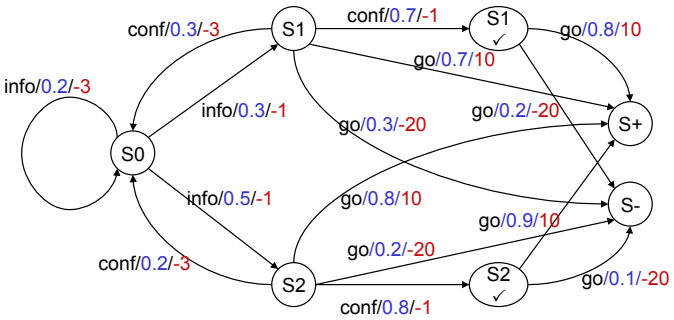
Language Technology II: Dialogue Learning II

Summer 2012

Manfred Pinkal

MDP





Expected cumulative reward and optimal policy

- The expected cumulative reward of taking action *a* in state s is defined by the "Bellman equation".

$$Q(s,a) = \sum_{s' \in S} P(s' | s,a) * [R(s',a) + \gamma * \max_{a' \in A} Q(s',a')]$$

- The optimal policy in a given state s is the one selecting the action which maximises the expected cumulative reward:

$$\pi^*(s) = \arg\max_{a' \in A} Q(s,a')$$

Estimating *P*

- Produce a dialogue corpus using WoZ experiments.
- Learn n-gram probabilities of state transitions (user dialogue moves) from the corpus.

Instantiating *R*

Options:

- Determine immediate reward by hand, using intuition and experience.
  - Arbitrary at least to some degree.
- Assess reward for the dialogue as a whole via (SASSI style) user questionnaires. Only final state-action pairs get non-zero reward.
  - Sparse data problem: Human users assess quality of the full dialogue sequence they have gone through; no assessment of sequences which have not occurred in the data.
- Approximate user assessment through measurable features.
  - PARADISE

The PARADISE questionnaire

- TTS Performance: Was the system easy to understand?
- ASR Performance: Did the system understand what you said?
- Task Ease: Was it easy to find the information you wanted?
- Interaction Pace: Was the pace of interaction with the system appropriate?
- User Expertise: Did you know what you could say at each point in the dialogue?
- System Response: How often was the system sluggish and slow to reply to you?
- Expected Behaviour: Did the system work the way you expected it to?
- Comparable Interface: How did the system's vioce interface compare to other systems?
- Future Use: From your current experience with using the system, do you think you would use the system regularly?

PARADISE: Details

- Training data: A set of dialogues (including log-files) produced by interaction of a dialogue system A with different subjects.
- Assessment of user satisfaction through questionnaire
  - User satisfaction := the arithmetic mean of numeric values assigned to the nine questions of the questionnaire
- Task success information:
  - Either $\in \{0, 1\}$, Succeed or Fail
  - Or $\in [0, 1]$, e.g., the proportion of appropriately filled slots (for information-seeking/form-filling dialogue)
  - Or some measure for the agreement between actual and correct slot fillers
- Indicators for dialogue costs:
  - Efficiency measures: Elapsed time, # of System turns, # of user turns
  - Qualitative measures: # of timeout prompts, # of rejects, # of helps, # of cancels, # of barge-ins, mean ASR score
- Compute the best fitting function from task success and dialogue cost information to satisfaction value via linear regression.

The Performance Function

$$US = (\alpha * N(\kappa)) - \sum_{i=1}^{n} w_i * N(c_i)$$

- κ is task success, c_i are the cost factors. *N* is normalisation function ($N(\kappa)$ and $N(c_i)$ normalised task success and cost factors, respectively.
- α and w_i are weights on κ and the c_i, determined by linear regression.

Determining the expected cumulative reward

$$Q(s,a) = \sum_{s' \in S} P(s' \mid s,a) * [R(s',a) + \gamma * \max_{a' \in A} Q(s',a')]$$

- Option 1: Compute the value of $Q(s,a)$ recursively from the Bellman equation for given P and R.
- Option 2: Approximate the optimal policy by carrying out dialogues with a simulated user. Several alternative algorithms are available. Among the most popular is Q-learning.

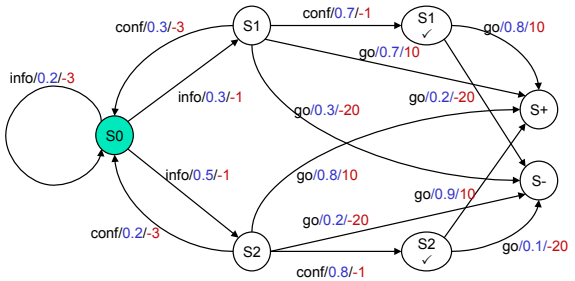
Q-Learning

- User simulation: Given state *s* and system action *a*, the environment generates state *s'* with probability *P(s'| a,s)* (state transition function).
- The system
  1. selects next action *a'* from state *s'* according to an intermediate estimate of the expected cumulative reward Q *(s', a')*, executes action, and
  2. updates the intermediate Q-value
- Intermediate Q-estimates for all state-action pairs are maintained in a "Q-table". The Q-table is initialized either randomly or with 0. Example:

	s1	s2	s3
a1	1.75	0.00	-7.62
a2	2.22	4.13	0.52
a3	-7.00	-5,35	8.75

Q-Learning Algorithm: An example

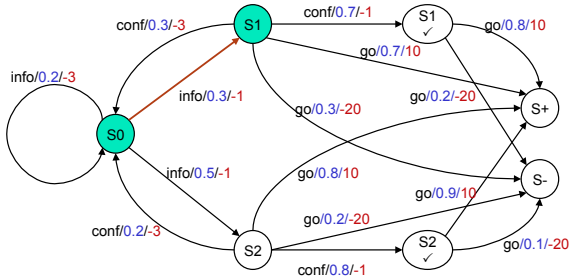
- Initiate Q-table with 0.
- Set s to S0
- Choose action from S0 (here, only 'info' is admissible)



	S0	S1	S1✓
info	0.00	0.00	0.00
conf	0.00	0.00	0.00
go	0.00	0.00	0.00

Example Cont'd

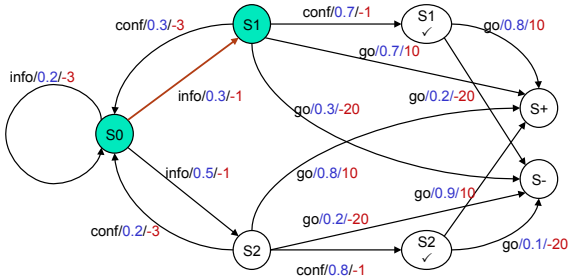
•Take the action, observe resulting state s' and reward r.
- In our example, s' happens to be S1 (chance of 30%):
- R(info, S1) = -1



	S0	S1	S1✓
info	0.00	0.00	0.00
conf	0.00	0.00	0.00
go	0.00	0.00	0.00

Example Cont'd

•Compute new Q-estimate for Q(S0, info)



	S0	S1	S1✓
info	0.00	0.00	0.00
conf	0.00	0.00	0.00
go	0.00	0.00	0.00

The *Q*–Learning Formula

- Bellmann Equation:

$$Q(s,a) = \sum_{s' \in S} P(s' \mid s,a) * [R(s',a) + \gamma * \max_{a' \in A} Q(s',a')]$$

- Estimate of Q for state s and action a:

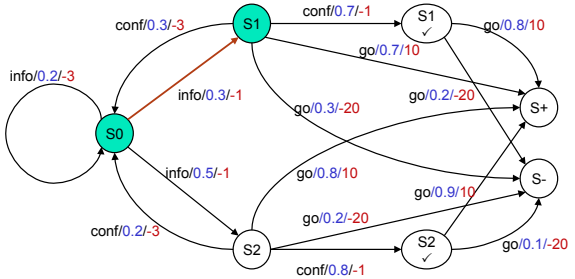
$$\underbrace{Q(s,a)}_{\text{new}} := (1-\alpha) * \underbrace{Q(s,a)}_{\text{old}} + \alpha * \left(R(s',a) + \gamma * \max_{a' \in A} Q(s',a') \right)$$

- The new Q-value estimate must be composed of the old value (representing the reward information accumulated so far), and the new estimate of the reward associated with transition to state s'. Parameter α (0≤α≤1) determines the step size.
- Learning usually starts with high step size (in the beginning, the Q-table is based on little or no evidence), which is then continuously reduced.

Example Cont'd

•Compute new Q-estimate for Q(S0, info)

$$Q(s,a) := Q(s,a) + \alpha * \left(r + \gamma * \max_{a' \in A} Q(s',a') - Q(s,a) \right)$$



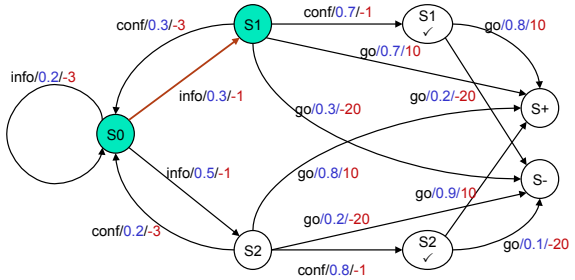
	S0	S1	S1✓
info	0.00	0.00	0.00
conf	0.00	0.00	0.00
go	0.00	0.00	0.00

Example Cont'd

•Compute new Q-estimate for Q(S0, info)

$$Q(S0,Info) := 0{,}5 * Q(S0,Info) + 0{,}5 * \left(-1 + \max\left[Q(S1,conf), Q(S1,go)\right]\right)$$

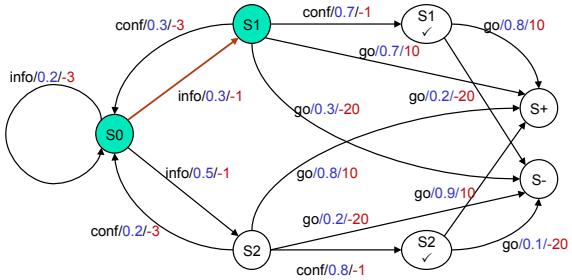
$$= 0 + 0{,}5 * \left(-1 + \max\left[0{,}0\right] - 0\right) = -0{,}5$$



	S0	S1	S1✓
info	0.00	0.00	0.00
conf	0.00	0.00	0.00
go	0.00	0.00	0.00

Example Cont'd

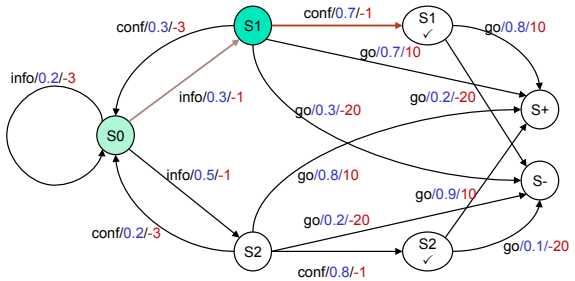
•Replace old Q-estimate with new Q-estimate



	S0	S1	S1✓
info	-0.5	0.00	0.00
conf	0.00	0.00	0.00
go	0.00	0.00	0.00

Example Cont'd

•Choose action from S1.

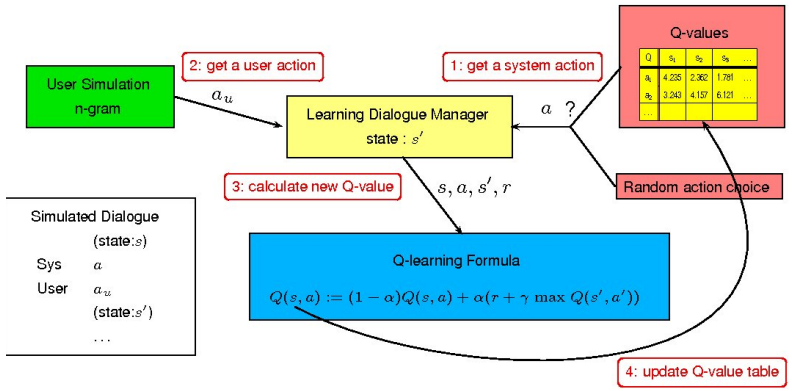


	S0	S1	S1✓
info	-0.5	0.00	0.00
conf	0.00	0.00	0.00
go	0.00	0.00	0.00

Action Selection

- If the Q-table would contain the real expected cumulative rewards, we would know the optimal policy and could always select the optimal action (that with the highest Q-value).
- But the Q-table contains temporary estimates. We therefore mix two selection modes:
  - Exploitation of the known Q-value estimates (which allows goal-directed, "greedy" search)
  - Exploration of new options via random selection of actions (which helps to find globally optimal solutions)
- An "exploration probability" ε can be given as a parameter, which determines the percentage to which the agent selects an action randomly. Two standard ways of handling ε:
  - Consistently diminishing ε over time.
  - Two phases: Exploration phase (high ε); optimization phase (low ε).

The Basic Q-learning Cycle

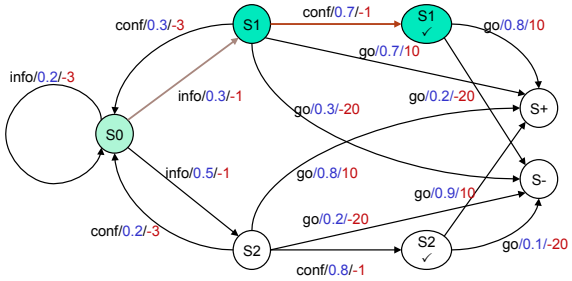


Example Cont'd

•Choose action from S1
- ➢ Action chosen is: *conf*

•Take the action, observe resulting state s' and reward R.
- ➢ s' happens to be S1✓ (chance of 70%), r= -1

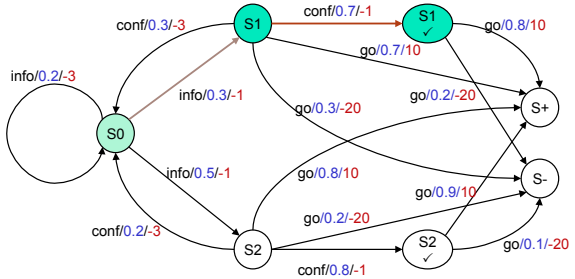
➢Compute new Q-estimate for Q(S1, conf).



	S0	S1	S1✓
info	-0.5	0.00	0.00
conf	0.00	0.00	0.00
go	0.00	0.00	0.00

Example Cont'd

$$Q(S1,conf) := 0{,}5 * Q(S1,conf) + 0{,}5 * \left(-1 + \max\left[Q(S1v,go)\right]\right)$$
$$= 0 + 0{,}5 * (-1 + 0) = -0{,}5$$



	S0	S1	S1✓
info	-0.5	0.00	0.00
conf	0.00	-0.5	0.00
go	0.00	0.00	0.00

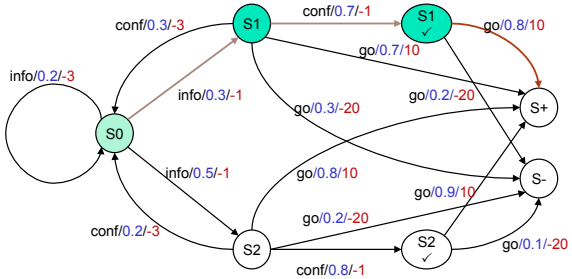
Example Cont'd

•Choose action from S2: *go*

•Take action, observe resulting state s'(S+) and reward r (+10).

• Copmpute new Q-estimate and replace old one.

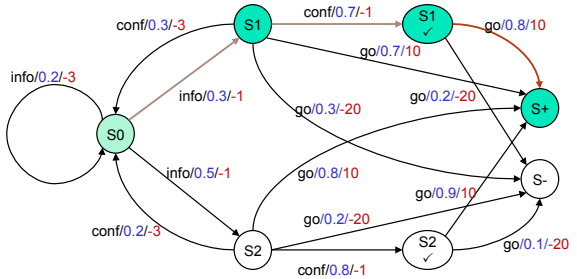
$$Q(S1v,go) := 0{,}5 * Q(S1v,go) + 0{,}5 * \left(-1 + \max[\varnothing]\right)$$
$$= 0 + 0{,}5 * (10 + 0) = 5$$



	S0	S1	S1✓
info	-0.5	0.00	0.00
conf	0.00	-0.5	0.00
go	0.00	0.00	+5.0

Example Cont'd

•If the second and third walk through the dialogue are identical to the first one (i.e., if the system selects the same actions, an the environment happens to react in the same way), the Q-table is updated as can be seen below.

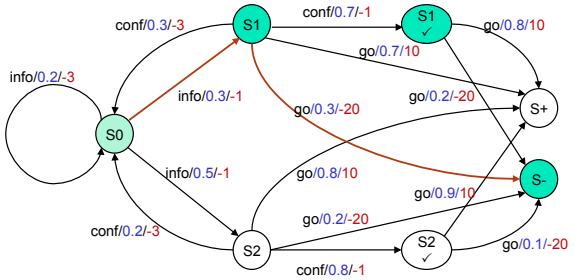


	S0	S1	S1✓
info	-0.75	0.00	0.00
conf	0.00	1.75	0.00
go	0.00	0.00	7.50

	S0	S1	S1✓
info	0.00	0.00	0.00
conf	0.00	4.13	0.00
go	0.00	0.00	8.75

Example Cont'd

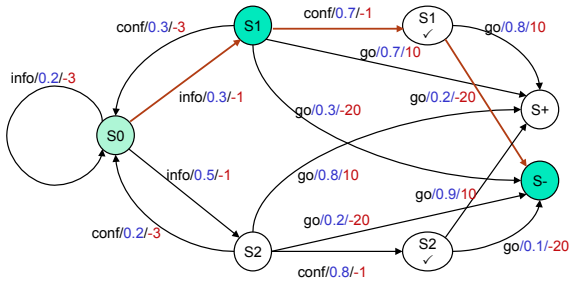
•Now, let the system try a direct, unconfirmed go:
the environments reaction reaction (ASR failure) may lead to a false destination trip of the elevator.



	S0	S1	S1✓
info	1.75	0.00	0.00
conf	0.00	4.13	0.00
go	0.00	-10	8.75

Example Cont'd

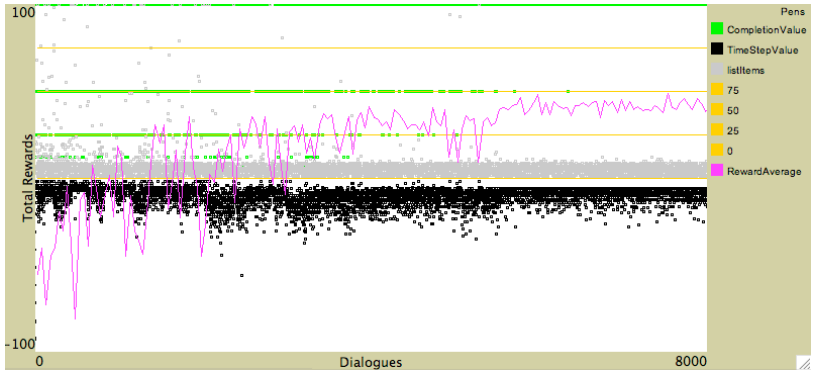
•But cautious version with confirmation may also fail ...



Simulation-based RL

- After a large number of iterations, the Q-table converges, and the optimal strategy can be read off.
- It is crucial that both action choice guided by the Q-table (greedy mode) and random action choice are combined.
- Typically, learning starts with an exploration phase (more random actions), and continues with an optimisation phase (selection by Q-table).

Example for a Policy Learning Curve



An experiment with 8,000 dialogues
(From Verena Rieser's Dissertation)

Simulation-based RL

- Advantages:
  - Allows exhaustive exploration of the space of possible policies
  - Allows exploration of completely novel strategies
  - Some global design decisions (state space, action set) can be changed without great effort.
- Problems:
  - Quality of learning strategy depends on quality of simulated environment.
  - Is the optimal strategy gained from simulation experiments also optimal for real users?

How realistic is RL?

- A general problem for all dialogue learning methods is the large state space (exponentially growing with number of features/dimensions)
- So far, RL methods are not straightforwardly applicable for the design of dialogue systems with realistic complexity.
- Methods to get around the complexity trap:
  - State space reduction
  - „Sub-Strategy Learning": Handcode most of the policy, and leave only special difficult design decisions for automatic optimisation