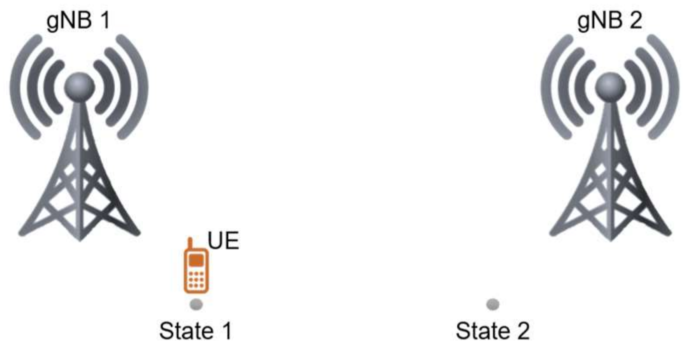
RL for 5G. Toy Example 1/4

- Consider a scenario with 2 gNBs, 1 UE
- States
  - UE1 moves between state 1 and state 2 probabilistically
  - Modeled as 2 state Markov chain
  - Transition probabilities (next slide)
- Actions
  - Association with gNB1 (A1) or with gNB2 (A2)
- Reward Matrix: Obtained using NetSim simulations
  - If UE1 associates with gNB1 in S1 it sees a throughput of 10 Mbps. Else, if it associates with gNB2 then throughput is 1 Mbps
  - If UE1 associates with gNB2 in S2 it sees a throughput of 10 Mbps. Else, if it associates with gNB1 then throughput is 1 Mbps
  - The reward matrix with elements R_{sa} is

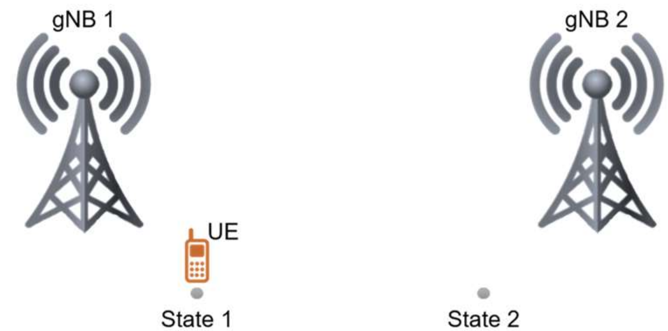
$$\begin{bmatrix} R_{11} & R_{12} \\ R_{21} & R_{22} \end{bmatrix} = \begin{bmatrix} 10 & 1 \\ 1 & 10 \end{bmatrix}$$



Q-Learning: Toy Example 1

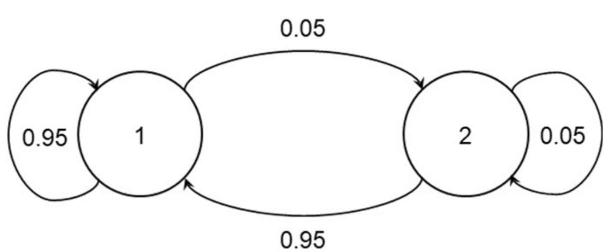
We set arbitrary state transition probabilities T (s’, s, a) as

- T[S[1], A[1], S[1]] = 0.95
- T[S[1], A[2], S[1]] = 0.05
- T[S[1], A[1], S[2]] = 0.05
- T[S[1], A[2], S[2]] = 0.95
- T[S[2], A[1], S[1]] = 0.05
- T[S[2], A[2], S[1]] = 0.95
- T[S[2], A[1], S[2]] = 0.95
- T[S[2], A[2], S[2]] = 0.05



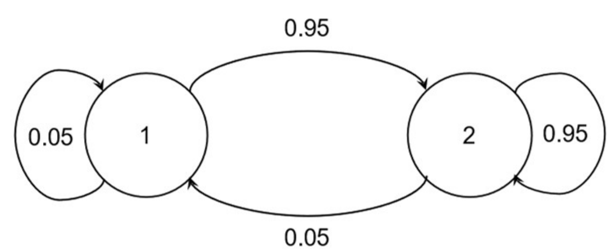
A1 Matrix

	S1	S2
S1	0.95	0.05
S2	0.5	0.5



A2 Matrix

	S1	S2
S1	0.5	0.5
S2	0.05	0.95



Q-Learning: A Toy Example 1

$$Q^{new}(s_t, a_t) \leftarrow \underbrace{Q(s_t, a_t)}_{\text{current value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \overbrace{\bigg(\underbrace{\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}}}_{\text{new value (temporal difference target)}} - \underbrace{Q(s_t, a_t)}_{\text{current value}} \bigg)}^{\text{temporal difference}}$$

Bellman's equations for the optimal value $V^*(s)$:

$$V^*(1) = \max_a (R(1,a) + \gamma\big(pV^*(1) + (1-p)V^*(2)\big)$$

$$V^*(2) = \max_a (R(2,a) + \gamma\big((1-p)V^*(1) + (p)V^*(2)\big)$$

To solve, we (i) substitute from the Reward matrix, and (ii) see that $V^*(1) = V^*(2)$ by symmetry. We get

$$V^*(1) = 10 + \gamma V^*(1)$$

For $\gamma = 0.9$, we get

$$V^*(1) = V^*(2) = \frac{10}{1-\gamma} = 100$$

Analytical calculation of the Q^*table

$$Q^*_{11} = \mathrm{R}_{11} + \gamma\big(pV^*(1) + (1-p)V^*(2)\big)$$

$$Q^*_{12} = R_{12} + \gamma\big(pV^*(1) + (1-p)V^*(2)\big)$$

And similarly, for $Q_{21}$and Q_{22}

We then get (for $\gamma = 0.9$)

$$Q = \begin{bmatrix} \frac{10}{1-\gamma} & \frac{1+9\gamma}{1-\gamma} \\ \frac{1+9\gamma}{1-\gamma} & \frac{10}{1-\gamma} \end{bmatrix} = \begin{bmatrix} 100 & 91 \\ 91 & 100 \end{bmatrix}$$

Thus, the optimal policy is given by

$$\pi^*(1) = \arg\max_a Q^*(1,a) = 1$$

$$\pi^*(2) = \arg\max_a Q^*(2,a) = 2$$

Toy Example 1: Julia code and output 4/4

```
using POMDPs, QMDP, POMDPModels, POMDPTools, QuickPOMDPs
using TabularTDLearning
using POMDPModels: TabularPOMDP
import Random
Random.seed!(1) # Seed the RNG for repeatability
# UE Count = 1 and eNB count = 2
S = [1, 2] # Two states - state 1 and state 2
A = [1, 2] # UE associates with eNB1 or eNB 2
T = zeros(2, 2, 2) # Initialize the 3D Transition probability Matrix
T[S[1], A[1], S[1]] = 0.95   # t.prob (s', a, s)
T[S[1], A[2], S[1]] = 0.05
T[S[1], A[1], S[2]] = 0.05
T[S[1], A[2], S[2]] = 0.95
T[S[2], A[1], S[1]] = 0.05
T[S[2], A[2], S[1]] = 0.95
T[S[2], A[1], S[2]] = 0.95
T[S[2], A[2], S[2]] = 0.05
R = zeros(2, 2) # Initialize the 2D Reward Matrix
R[S[1], A[1]] = 10  # R(s, a)
R[S[1], A[2]] = 1
R[S[2], A[1]] = 1
R[S[2], A[2]] = 10
Discount_factor = 0.9
mdp = TabularMDP(T, R, Discount_factor);
# use Q-Learning
exppolicy = EpsGreedyPolicy(mdp, 0.02)
q_learning_solver = QLearningSolver(exploration_policy=exppolicy,
                                    n_episodes=1,
                                    max_episode_length=10000,
                                    learning_rate=0.5,
                                    eval_every=10000,
                                    n_eval_traj=20,
                                    verbose=true);
policy = solve(q_learning_solver, mdp)
print("\nThe value table is:\n")
show(policy.value_table)
print("\nThe optimal action for all states are (shown as (s -> a)):\n")
showpolicy(mdp, policy)
```

- Code written in Julia using POMDPs and TabularTDLearning
- T is a 3D matrix of the form T(s', a ,s)
- R is a 2D matrix of the form R(s,a)
- Program output

  The value table is:
  [99.99 90.99;
  90.91 99.99]
  The optimal action for all states are (shown as (s -> a)):
  1 -> 1
  2 -> 2

- Output matches analysis (in previous slide)
- Self contained code: Can copy and paste it into VSCode or REPL