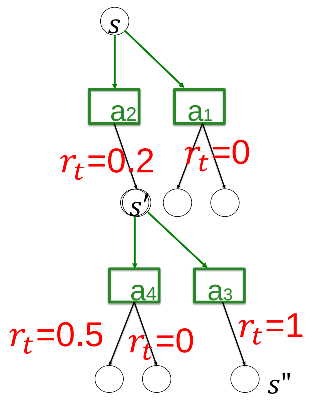
Artificial Neural Networks (Gerstner). Solutions for week 9

Variations of SARSA and continuous space

Exercise 1. Monte-Carlo Batch versus Bootstrap Batch

example trials:
1: **s,a2→** r=0.2,s',a4→ r=0
2: s',a3→ r=1
3: s',a4→ r=0
4: s',a3→ r=1
5: **s,a1** → r=0
6: s',a4→ r=0
7: s',a4→ r=0.5
8: s',a3→ r=1
9. **s,a2→** r=0.2,s',a4→ r=0.5
10. **s,a1** → r=0



The aim is to estimate Q-values on the graph as shown in the figure on top. We run 10 exploration trials with the rewards as indicated in the figure. Some trials start in state s others in state s'.

a. Use Monte-Carlo in batch version to calculate the cumulative expected reward $R(s,a)$ by averaging along all trials that start at s,a (discount factor $\gamma = 1$) for all state-action pairs.

b. Use Q-learning in batch version (= dynamic programming) to evaluate the estimated Q-values $Q(s,a)$ by exploiting the Bellman equation.

   Hint: Start at the terminal state and work up backwards.

c. Use Q-learning in online version to evaluate the estimated Q-values $Q(s,a)$

d. Which of the algorithms performs best on this example?

Solution:

trial(s)	R (Monte Carlo)	Q (Q-Learning batch)
2, 4, 8	$R(s',a_3) = \frac{1}{3}\cdot(1+1+1)$	$Q(s',a_3) = \frac{1}{3}\cdot(1+1+1)$
1, 3, 6, 7, 9	$R(s',a_4) = \frac{1}{5}\cdot(0.5+0.5)$	$Q(s',a_4) = \frac{1}{5}\cdot(0.5+0.5)$
5, 10	$R(s,a_1) = 0$	$Q(s,a_1) = 0$
1, 9	$R(s,a_2) = \frac{1}{2}\cdot(2\cdot 0.2+0.5)$	$Q(s,a_2) = r + \max_a Q(s',a) = 0.2 + 1$

trial	Q (Q-Learning online)
1	$Q(s, a_2) = \alpha(0.2 + 0)$
	$Q(s', a_4) = 0$
2	$Q(s', a_3) = \alpha \cdot 1$
3	$Q(s', a_4) = 0$
4	$Q(s', a_3) = \alpha \cdot 1 + \alpha(1 - \alpha) = 2\alpha - \alpha^2$
5	$Q(s, a_1) = 0$
6	$Q(s', a_4) = 0$
7	$Q(s', a_4) = \alpha \cdot 0.5$
8	$Q(s', a_3) = 2\alpha - \alpha^2 + \alpha(1 - 2\alpha + \alpha^2) = 3\alpha - 2\alpha^2 + \alpha^3$
9	$Q(s, a_2) = 0.2\alpha + \alpha(0.2 + 3\alpha - 2\alpha^2 + \alpha^3 - 0.2\alpha) = 0.4\alpha + 2.8\alpha^2 - 2\alpha^3 + \alpha^4$
	$Q(s', a_4) = 0.5\alpha + \alpha(0.5 - 0.5\alpha) = \alpha - 0.5\alpha^2$
10	$Q(s, a_1) = 0$

Q-learning in batch version is better. Online Q-learning is typically slower than batch Q-learning as it goes forward through the graph. The batch version can better propagate rewards as it goes backward through the graph.

Exercise 2. Q-values for continuous states

We approximate the state-action value function $Q(s, a)$ by a weighted sum of basis functions (BF):

$$Q(s, \tilde{a}) = \sum_j w_{\tilde{a}j} \Phi(s - s_j) \, ,$$

where $\Phi(x)$ is the BF "shape", and the s_j's represent the centers of the BFs.

Calculate

$$\frac{dQ(s, a)}{dw_{ai}} \, ,$$

the gradient of $Q(s, a)$ along w_{ai} for a specific weight linking the basis function i to the action a.

Solution:

Using the definition of $Q(s, a)$ given, we find the gradient:

$$\frac{dQ(s, a)}{dw_{aj}} = \Phi(s - s_j) \, .$$

Therefore the direction of the gradient vector (dQ/dw_{aj}) for $j = 1, \ldots, K$ is given by the magnitude of responses $(\Phi(s - s_j))$ of all basis functions.

Exercise 3. Eligibility traces

In week 7 we applied, in the exercises, the SARSA algorithm to the case of a linear track with actions 'up' and 'down'. We found that it takes a long time to propagate the reward information through state space. The eligibility trace is introduced as a solution to this problem.

Reconsider the exercise from week 7, but include an eligibility trace: for each state s and action a, a memory $e(s,a)$ is stored. At each time step, all the memories are reduced by a factor λ: $e(s,a) = \lambda e(s,a)$, except for the memory corresponding to the current state s^* and action a^*, which is incremented:

$$e(s^*,a^*) = \lambda e(s^*,a^*) + 1\,. \tag{1}$$

Now, unlike the case without eligibility trace, all Q-values are updated at each time step according to the rule

$$\forall (s,a) \quad \Delta Q(s,a) = \eta\left[r - (Q(s^*,a^*) - Q(s',a'))\right] e(s,a)\,. \tag{2}$$

where s^*, a^* are the current state and action, and s', a' are the immediately following state and action.

We want to check whether the information about the reward propagates more rapidly. To find out, assume that the agent goes straight down in the first trial. In the second trial it uses a greedy policy. Calculate the Q-values after two complete trials and report the result.

Hint: Reset the eligibility trace to zero at the beginning of each trial.

Solution:

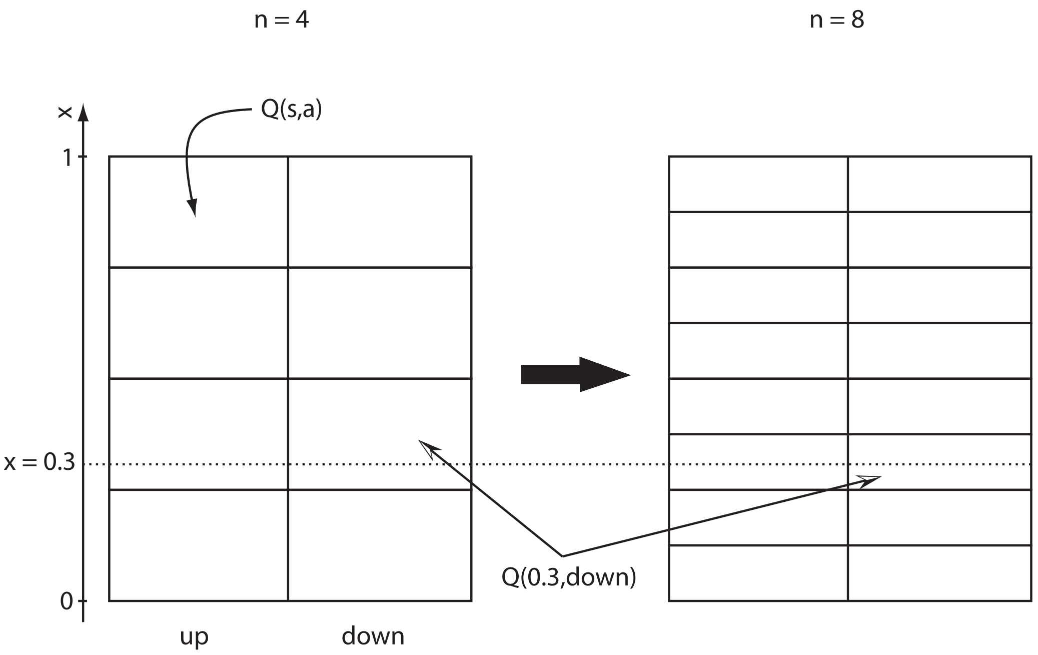
The table below shows the evolution of the Q values for each relevant state action pair during the first 2 trials, starting at the first step when there is a non-zero update $\Delta = \eta\left[r - (Q(s^*,a^*) - Q(s',a'))\right]$. We assume that the agent goes straight down in the first trial, that it always picks the best action, and that the eligibility traces are reset when the agent picks the reward, and the agent is put back to the starting position.

trial	transition		(s,a_1)	(s',a_1)	(s'',a_1)	Δ
1	$s'' \to s'''$	Q	0	0	0	η
		e	λ^2	λ	1	$-$
2	$s \to s'$	Q	$\eta\lambda^2$	$\eta\lambda$	η	$\eta^2(\lambda-\lambda^2)$
		e	1	0	0	$-$
2	$s' \to s''$	Q	$\eta\lambda^2 + \eta^2(\lambda-\lambda^2)$	$\eta\lambda$	η	$\eta^2(1-\lambda)$
		e	λ	1	0	$-$
2	$s'' \to s'''$	Q	$\eta\lambda^2 + 2\eta^2\lambda - 2\eta^2\lambda^2$	$\eta\lambda + \eta^2(1-\lambda)$	η	$\eta - \eta^2$
		e	λ^2	λ	1	$-$
3	$s \to s'$	Q	$2\eta\lambda^2 + 2\eta^2\lambda - 3\eta^2\lambda^2$	$2\eta\lambda + \eta^2 - 2\eta^2\lambda$	$2\eta - \eta^2$	$\cdots$
		e	1	0	0	$-$

$\cdots$

Although the Q-values for s'' are the same as without the eligibility trace (see exercise from last week), the Q-values for s and s' already start to approach their asymptotical value (i. e. 1) in the first trial.

Exercise 4. Eligibility traces in continuous space



The left part of the figure above shows a different representation of last week's "linear track" exercise: the vertical divisions represent different states, and the two column correspond to the two possible actions available to the agent: go up or down. Each square represents a possible state-action combination, and thus a Q-value. (Note that the uppermost "up" action and the lowermost "down" action should be "greyed out": they are impossible. But this is not relevant to the rest of this exercise.)

Suppose now that the agent moves in a continuous 1-dimensional space $0 \leq x \leq 1$, with the target located at $x = 0$. Separate this state space into n equal bins of width $\Delta x = 1/n$. In each time step, the agent moves by one bin. Vary the discretization by varying n: $n = 4, 8, 16\ldots$

a. Suppose that one action (such as move down) corresponds to one time step Δt in 'real time'. How should we rescale the parameter Δt, so that the speed $v = \Delta x/\Delta t$ remains constant when we change the discretization?

b. We use an eligibility trace with decay parameter λ. How should we rescale λ, in order that the "speed of information propagation" in SARSA(λ) remains constant?

   Hint: Consider for example, the Q-value at $x = 0.5$ after 2 complete learning trials.

Solution:

a. If $\Delta x/\Delta t$ has to remain constant, then Δt should vary like Δx, i.e., $\Delta t \propto 1/n$.

b. A point "sitting" at x is $x/\Delta x = x \cdot n$ steps away from the reward (assuming we always choose the "down" action). As we have seen in exercise 2, on the first trial, the Q-value of a state d steps from the trial is updated proportional to λ^d. Thus, if we want the update to stay constant under rescaling, we need

$$\Delta Q(s,a) \propto \lambda^d = \lambda^{x \cdot n} = cst$$

   This holds if we replace λ by a "rescaled" $\tilde{\lambda} = \lambda^{\frac{1}{n}}$.

Exercise 5. Gradient-based learning of Q-values

Assume again that the Q-values are expressed as a weighted sum of 400 basis functions: $Q(s,a) = \sum_{k=1}^{400} w_{ak}\Phi(s - s_k)$. For the moment the function Φ is arbitrary, but you may think of it as a Gaussian function. Note that s and s_k are usually vectors in $\mathbb{R}^N$ in this case. There are 3 different actions so that the total number of weights is 1200. Now consider the error function $E_t = \frac{1}{2}\delta_t^2$, where

$$\delta_t = r_t + \gamma \cdot Q(s', a') - Q(s,a) \tag{3}$$

is the reward prediction error. Our aim is to optimize $Q(s,a)$ by changing the parameters w.

a. Find a learning rule that minimizes the error function E_t by gradient decent. Consider the case where the actions a and a' are different.

   Write down the learning rule. How many weights need to be updated in each time step?

b. Find a learning rule that minimizes the error function E_t by gradient decent. Consider the case where the actions a and a' are the same.

   Write down the learning rule.

   Is there any difference to the case considered in a?

c. Suppose that the input space is two-dimensional and you discretize the input in 400 small square 'boxes' (i.e., 20×20). The basis function $\Phi(s - s_k)$ is now the indicator function: it has a value equal to one if the current state s is in 'box' k and zero otherwise.

   How do your results from (a) and (b) look like in this case?

d. The learning rule in c is very similar to standard SARSA. What is the difference?

e. Assume that $Q(s', a')$ in Equation 3 does not depend on the weights. How does the learning rule look like for a, b and c. How is your result related to standard discrete SARSA?

Solution:

a. Let's start by computing the derivative of $Q(s,a)$ with respect to $w_{\tilde{k}}^{\tilde{a}}$ (we'll use this later):

$$\frac{\mathrm{d}Q(s,a)}{\mathrm{d}w_{\tilde{k}}^{\tilde{a}}} = \delta_{a\tilde{a}}\,\Phi(s - s_{\tilde{k}})\,,$$

   where $\delta_{a\tilde{a}}$ is the Kroneker, i.e., it is 1 if $a = \tilde{a}$, and 0 otherwise (not to be confused with δ_t).

   We then compute the gradient, i.e., the derivative of E_t with respect to $w_{\tilde{k}}^{\tilde{a}}$, using the chain rule a few times and the result above:

$$\begin{aligned}\frac{\mathrm{d}E_t}{\mathrm{d}w_{\tilde{k}}^{\tilde{a}}} &= \delta_t\left[\gamma\frac{\mathrm{d}Q(s',a')}{\mathrm{d}w_{\tilde{k}}^{\tilde{a}}} - \frac{\mathrm{d}Q(s,a)}{\mathrm{d}w_{\tilde{k}}^{\tilde{a}}}\right] \\ &= \delta_t\left[\gamma\delta_{a'\tilde{a}}\,\Phi(s' - s_{\tilde{k}}) - \delta_{a\tilde{a}}\,\Phi(s - s_{\tilde{k}})\right].\end{aligned}$$

   To turn this into a learning rule, we have to move the weights in the direction that *minimizes* the error, i.e.

$$\Delta w_{\tilde{k}}^{\tilde{a}} = -\eta\frac{\mathrm{d}E_t}{\mathrm{d}w_{\tilde{k}}^{\tilde{a}}} = \eta\,\delta_t\left[\delta_{a\tilde{a}}\,\Phi(s - s_{\tilde{k}}) - \gamma\delta_{a'\tilde{a}}\,\Phi(s' - s_{\tilde{k}})\right]. \tag{4}$$

   $2 \cdot 400$ weights (for actions a and a') need to be updated in each step.

b. In the case where $a = a'$ (i.e., the action taken is the same in the two consecutive steps):

$$\Delta w_{\tilde{k}}^{\tilde{a}} = \eta\, \delta_t\ (\Phi(s - s_{\tilde{k}}) - \gamma \Phi(s' - s_{\tilde{k}}))\, \delta_{a\tilde{a}}. \tag{5}$$

400 weights need to be updated.

c. The learning rules look the same. Only 2 weights need to be updated if $a \neq a'$ and 1 weight if $a = a'$, because $\Phi(s - s_k) = 0$ if s does not fall into the kth box.

d. Without the term $\gamma \Phi(s' - s_{\tilde{k}})$, the algorithm becomes identical to standard SARSA.

e. If $Q(s', a')$ is a fixed target that does not depend on the weights, the term $\gamma \Phi(s' - s_{\tilde{k}})$ drops out of the learning rules and with the small square box function approximator the algorithm becomes equivalent to SARSA. This is sometimes called semigradient learning.