1 Functional Approximation

46. Autonomous Orchard

$$\vec{w} = \begin{pmatrix} w_1 \\ w_2 \\ w_3 \end{pmatrix}, \vec{S} = \begin{pmatrix} A \\ B \\ C \end{pmatrix},$$
$$\hat{v}(\vec{w}, \vec{S}) = w_1 \cdot A + w_2 \cdot B + w_3 \cdot C,$$
$$\nabla \hat{v}(\vec{w}, \vec{S}) = \begin{pmatrix} A \\ B \\ C \end{pmatrix}$$

1^{st} step:

$$\begin{aligned} w_1 &= w_0 + \alpha[G - \hat{v}(\vec{w_0}, \vec{S_0})] \nabla \hat{v}(\vec{w_0}, \vec{S_0}) \\ &= \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} + 0.01[3 - (0 \cdot 4 + 0 \cdot 7 + 0 \cdot 1)] \begin{pmatrix} 4 \\ 7 \\ 1 \end{pmatrix} \\ &= \begin{pmatrix} 0.12 \\ 0.21 \\ 0.03 \end{pmatrix} \end{aligned}$$

2^{nd} step:

$$\begin{aligned} w_2 &= w_1 + \alpha[G - \hat{v}(\vec{w_1}, \vec{S_1})] \nabla \hat{v}(\vec{w_1}, \vec{S_1}) \\ &= \begin{pmatrix} 0.12 \\ 0.21 \\ 0.03 \end{pmatrix} + 0.01[-15 - (0.12 \cdot 10 + 0.21 \cdot 6 + 0.03 \cdot 0)] \begin{pmatrix} 10 \\ 6 \\ 0 \end{pmatrix} \\ &= \begin{pmatrix} -1.626 \\ -0.838 \\ 0.03 \end{pmatrix} \end{aligned}$$

3^{rd} step:

$$\begin{aligned} w_3 &= w_2 + \alpha[G - \hat{v}(\vec{w_2}, \vec{S_2})] \nabla \hat{v}(\vec{w_2}, \vec{S_2}) \\ &= \begin{pmatrix} -1.626 \\ -0.838 \\ 0.03 \end{pmatrix} + 0.01[5 - -(1.626 \cdot 20 + (-0.838) \cdot 1 + 0.03 \cdot 15)] \begin{pmatrix} 10 \\ 6 \\ 0 \end{pmatrix} \\ &= \begin{pmatrix} 5.956 \\ -0.459 \\ 5.716 \end{pmatrix} \end{aligned}$$

4^{th} step:

$$\begin{aligned} w_4 &= w_3 + \alpha[G - \hat{v}(\vec{w_3}, \vec{S_3})] \nabla \hat{v}(\vec{w_3}, \vec{S_3}) \\ &= \begin{pmatrix} 5.956 \\ -0.459 \\ 5.716 \end{pmatrix} + 0.01[21 - (5.956 \cdot 4 + (-0.459) \cdot 19 + 5.716 \cdot 3)] \begin{pmatrix} 4 \\ 19 \\ 3 \end{pmatrix} \\ &= \begin{pmatrix} 5.505 \\ -2.6 \\ 5.38 \end{pmatrix} \end{aligned}$$

Additional to the calculations by hand, they were implemented as the Python script `autonomous_orchard.py` to cross-check the results.

47. Exercise 9.1 Show that tabular methods such as presented in Part I of this book are a special case of linear function approximation. What would the feature vectors be?

48. Exercise 10.4 Give pseudocode for a differential version of semi-gradient Q-learning.

```
Algorithm 1: Differential semi-gradient Q-learning
Initialize: $\vec{w}, \vec{R}, S$
for each step do
    choose $A$ as a function of $\hat{q}(S, \cdot, \vec{w})$ (e.g. $\epsilon$ -greedy)
    take action $A$ and observe $R, S'$
    $\delta \leftarrow R - \overline{R} + max_a \hat{q}(S', a, \vec{w}) - \hat{q}(S, A, \vec{w})$
    $\vec{R} \leftarrow \vec{R} + \beta\delta$
    $\vec{w} \leftarrow \vec{w} + \alpha\delta\nabla q(S, A, \vec{w})$
    $S \leftarrow S'$
end
```

49. Exercise 11.1 Convert the equation of n-step off-policy TD (7.9) to semi-gradient form. Give accompanying definitions of the return for both the episodic and continuing cases.

Using the definitions:

$$G_{t:t+n} \doteq R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n \hat{v}(S_{t+n}, w_{t+n-1})$$

for the episodic case and

$$G_{t:t+n} \doteq R_{t+1} - R_t + R_{t+2} - \overline{R}_{t+1} + \cdots + R_{t+n} - \overline{R}_{t+n-1} + \hat{v}(S_{t+1}, w_{t+n-1})$$

for the continuous case, one can write the n-step off-policy TD in semi-gradient form as

$$w_{t+n} = w_{t+n-1} + \alpha\rho_{t:t+n-1}[G_{t:t+n} - \hat{v}(S_t, w_{t+n-1})]\nabla\hat{v}(S_t, w_{t+n-1})$$

50. Exercise 12.1 Just as the return can be written recursively in terms of the first reward and itself one-step later (3.9), so can the λ-return. Derive the analogous recursive relationship from (12.2) and (12.1).

51. Implementation Task: Reinforce

See Exercise 52 for result figurej and file `reinforce.py` for the implementation.

52. Implementation Task: Reinforce with Baseline

As environment to run the implemented REINFORCE algorithm against, the CartPole-v0 environment of the OpenAI gym framework was chosen.
Strangely, the variant with baseline performs worse than the variant without baseline. The used approximation for the value-function was a simple pre-calculated array containing the the discounted values of the states at certain time steps. This could be done, as per default, the maximum number of time steps of this environment is 200.

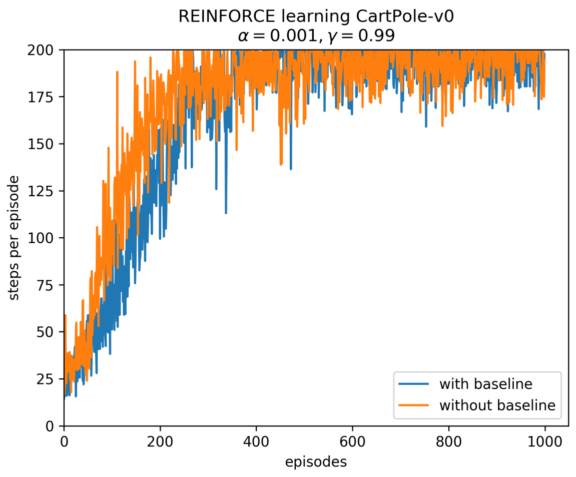

Figure 1: Comparison of steps taken per episode for learning the cartpole-v0 problem from the OpenAI gym environment using the REINFORCE algorithm once with baseline and without baseline.The shown results are averaged over 5 runs.

Results were generated using the file `reinforce.py`.

53. Implementation Task: Actor-Critic

Not implemented.

54. Implementation Task: Step-Size Parameter

To check what happens, when the condition $\sum_{n=1}^{\infty} = \alpha_n^2 < \infty$, the SARSA algorithm and the WindyGridWorld environment implemented in the previous exercise set were used. The program was run once with the step size α set to 0.5 and once to 1.5. On can see in Figure 2, that if $\alpha = 0.5$, the algorithm rather quickly converges, but if the step size is increased to 1.5, it never converges. The shown figure only goes up to 1000 episodes, but simulations were run longer and no change to the behavior was observed.

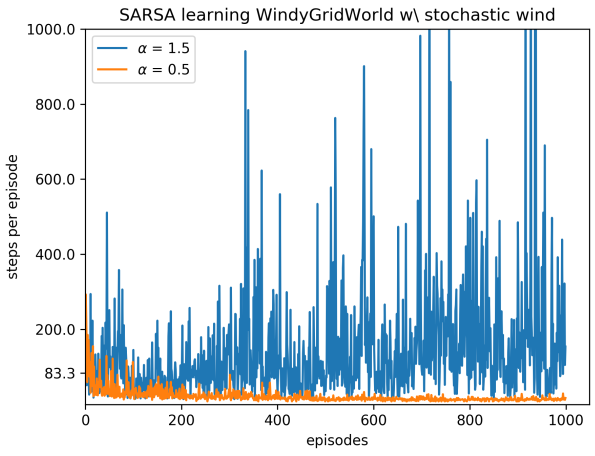

Figure 2: Comparison of steps taken per episode for learning the cartpole-v0 problem from the OpenAI gym environment using the REINFORCE algorithm once with baseline and without baseline.