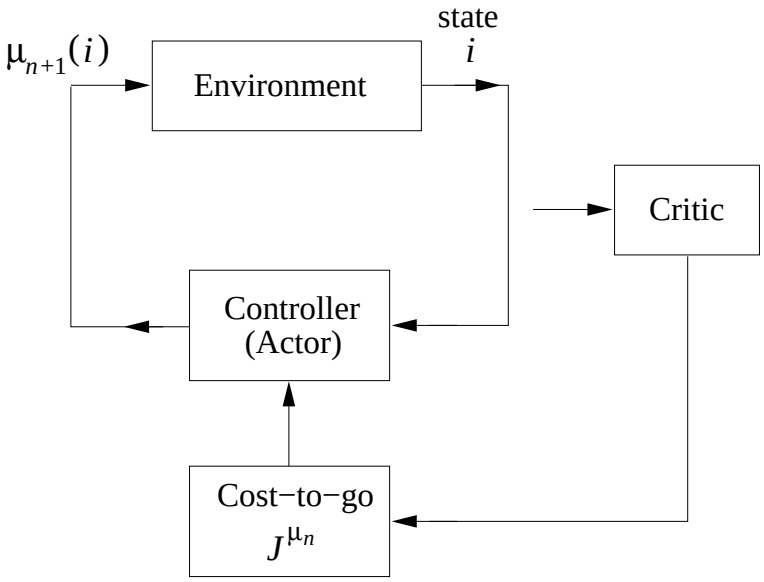
T-61.5030 Advanced course in neural computing

Solutions for exercise 10

1. The figure below presents an interesting interpretation of the policy iteration algorithm.



   In this interpretation, the policy evaluation step is viewed as the work of a *critic* that evaluates the performance of the current policy; that is, it calculates an estimate of the cost-to-go function J^{μ_n}. The policy improvement step is viewed as the work of a *controller* or *actor* that accounts for the latest evaluation node by the critic and acts out the improved policy μ_{n+1}. In short, the critic looks after policy evaluation and the controller (actor) looks after policy improvement, and the interaction between them goes on.

2. The value iteration algorithm requires knowledge of the state transition probabilities. In contrast, Q-learning operates without this knowledge. But through an iterative process, Q-learning learns estimates of the transition probabilities in an implicit manner. Recognizing the intimate relationship between value iteration and Q-learning, we may therefore view Q-learning as an adaptive version of the value iteration algorithm.

3. The neuronal filter in Figure 13.9 is a block which is used in the network of Figure 13.10. The output of the filter is $y(n) = \varphi(v(n))$, where $v(n)$ is the activation

$$v(n) = b + \sum_{l=0}^{p} w(l)x(n-l)\,.$$

   (Notice that for convenience we use notation which is simpler than that in Haykin's book.) The least mean square (LMS) algorithm aims at minimising the mean square error

$$\mathcal{E} = \sum_n \mathcal{E}(n) = \frac{1}{2}\sum_n e(n)^2\,,$$

where $e(n) = d(n) - y(n)$ is the error made by the network. For deriving an on-line learning algorithm we use the stochastic gradient descent method where one proceeds in the direction of the negative gradient of the instantaneous error $\mathcal{E}(n)$.

From the chain rule we have

$$\frac{\partial \mathcal{E}(n)}{\partial w_l} = \frac{\partial \mathcal{E}(n)}{\partial e(n)} \frac{\partial e(n)}{\partial y(n)} \frac{\partial y(n)}{\partial v(n)} \frac{\partial v(n)}{\partial w(l)} = e(n)(-1)\varphi'(v(n))x(n-l)\,.$$

The learning rule for adjusting the weight $w(l)$ is thus $\Delta w(l) = -\eta \partial \mathcal{E}(n)/\partial w(l) = \eta e(n)\varphi'(v(n))x(n-l)$, where η is the learning rate. A similar derivation for the bias b will give the learning rule $\Delta b = -\eta \partial \mathcal{E}(n)/\partial b = \eta e(n)\varphi'(v(n))$.

4. Once the structure is fixed, the parameters can be learned for instance by temporal back-propagation. For the design of the structure there are no general rules. Cross-validation, Bayesian models selection etc. can be used but the structure of the delay line has very many degrees of freedom.

   The available prior knowledge about the plausible time lags should be used for choosing the lengths of the delays, possible averaging lengths and so on. Often it is reasonable to have a higher resolution for the immediate past and increasingly coarse resolution for more distant past.

5. Let $\mathbf{x}(n) = [x(n)x(n-1)\ldots x(n-p)]$ denote the vector of current and delayed input signals applied to the distributed TLFN at time n. Let $y(n)$ denote the actual response of the network trained using the temporal back-propagation algorithm. With one-step prediction as the requirement, the desired response is the value of the input signal one step into the future, that is, $x(n+1)$. With these provisions, we may then proceed with the temporal back-propagation algorithm as described in Haykin, Section 13.9.