**T-61.5030 Advanced course in neural computing**

**Solutions for exercise 11**

1. The network in Figure 15.4 has the following equations $\mathbf{x}_I(n+1) = \mathbf{f}_1(\mathbf{x}_I(n), \mathbf{u}(n))$, $\mathbf{x}_{II}(n+1) = \mathbf{f}_2(\mathbf{x}_{II}(n), \mathbf{x}_I(n+1))$ and $\mathbf{x}_o(n+1) = \mathbf{f}_3(\mathbf{x}_o(n), \mathbf{x}_{II}(n+1))$, where we have denoted the feedforward functions of different layers by $\mathbf{f}_1$, $\mathbf{f}_2$ and $\mathbf{f}_3$. The output of the network is $\mathbf{y}(n) = \mathbf{x}_o(n+1)$. Therefore we have

$$\mathbf{x}_{II}(n+1) = \mathbf{f}_2(\mathbf{x}_{II}(n), \mathbf{f}_1(\mathbf{x}_I(n), \mathbf{u}(n)))$$

   and

$$\mathbf{x}_o(n+1) = \mathbf{f}_3(\mathbf{x}_o(n), \mathbf{f}_2(\mathbf{x}_{II}(n), \mathbf{f}_1(\mathbf{x}_I(n), \mathbf{u}(n)))) \, .$$

   If we now set

$$\mathbf{x}(n) = \begin{bmatrix} \mathbf{x}_I(n) \\ \mathbf{x}_{II}(n) \\ \mathbf{x}_o(n) \end{bmatrix} ,$$

$$\mathbf{f}(\mathbf{x}(n), \mathbf{u}(n)) = \begin{bmatrix} \mathbf{f}_1(\mathbf{x}_I(n), \mathbf{u}(n)) \\ \mathbf{f}_2(\mathbf{x}_{II}(n), \mathbf{f}_1(\mathbf{x}_I(n), \mathbf{u}(n))) \\ \mathbf{f}_3(\mathbf{x}_o(n), \mathbf{f}_2(\mathbf{x}_{II}(n), \mathbf{f}_1(\mathbf{x}_I(n), \mathbf{u}(n)))) \end{bmatrix}$$

   and

$$\mathbf{g}(\mathbf{x}(n), \mathbf{u}(n)) = \mathbf{f}_3(\mathbf{x}_o(n), \mathbf{f}_2(\mathbf{x}_{II}(n), \mathbf{f}_1(\mathbf{x}_I(n), \mathbf{u}(n)))) \, ,$$

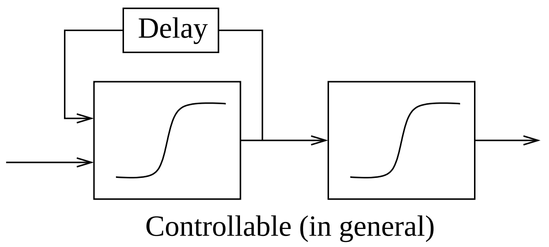
   we see that $\mathbf{x}(n+1) = \mathbf{f}(\mathbf{x}(n), \mathbf{u}(n))$ and $\mathbf{y}(n) = \mathbf{g}(\mathbf{x}(n), \mathbf{u}(n))$.

2. The internal state of the NARX model can be deduced by observing as many outputs as the delay line has delays. In other words, NARX model is always observable. This means that a state space model which is not observable cannot be modelled by a NARX model. Any observable state space model can be modelled by a NARX model, however. The delay line needs to be at least as long as the number of observations which are needed for deducing the internal state of the state space model.

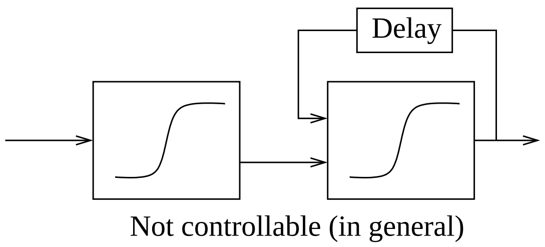
   The transformation of a NARX model into a state space is trivial and can be done using the same guidelines as were used in the previous problem (the internal state is the contents of the memory).

3. Any NARX model serves as an example of an observable network. The network in Figure 15.4. which was referred to in the problem 1. is an example of a network which is not, in general, observable because it has feedback from somewhere else than the outputs or inputs (which are assumed to be observed by the definition of observability).

   The following figure gives an example of networks which are (in general) controllable and uncontrollable. The layers of the networks (denoted by boxes with sigmoid) are assumed to compute weighted sums followed by saturating nonlinearities.



Controllable (in general)



Not controllable (in general)

The reason for the first network to be controllable is that the state of the network can always be overridden by suitable values for the inputs. This is not the case in the second network because the effect of the inputs is restricted by the saturating nonlinearity applied before the inputs can affect the layer which gets input from the internal state (the delayed values).

4. (a) Both algorithms compute the instantaneous gradient $\nabla_{\mathbf{w}}\mathcal{E}(n)$. RTRL achieves this by forward-propagating the derivatives of the state variables w.r.t. the weights, i.e., by updating the Jacobian matrix of states w.r.t. the weights. This means that in RTRL, backpropagation only needs to be done up to the inputs and states and the gradient is not propagated through the delays. In BPTT, the forward propagation is not done for Jacobian matrix and the gradient needs to be propagated through delays. In other words, most of the work in RTRL is done in the forward propagation phase while in BPTT the work is done in the back propagation phase.

   (b) In RTRL, the approximation of the instantaneous gradient $\nabla_{\mathbf{w}}\mathcal{E}(n)$ does not take into account the fact that changing weights affects the gradient earlier in time. This may cause instabilities.

   (c) The update has to change weights so little that the learning remains stable. By choosing a small enough learning rate, the time scale of the weight changes can be kept smaller than the time scale of the recurrent network operation.

5. The states are denoted by A and B and the input can be either zero or one. The automaton is supposed to tell whether a sequence contains an odd number of ones. The state changes each time the input is one and remains constant when the input is zero. If the automaton starts at state A, it is easy to see that number of ones is odd if at the end of the sequence the state is B.

   The conversion from finite state automaton to second order recurrent network is easy if the states and inputs are encoded by place-code such that A = (1, 0) and B = (0, 1) and similarly, zero = (1, 0) and one = (0, 1). The network gets two inputs and has two hidden units that each compute one output. Denote the two inputs by $u_1$ and $u_2$ and the outputs by $x_1$ and $x_2$. The input $v_k$ to hidden unit $k$, $k = 1, 2$, is a weighted sum of the products between state $x_i$ and input $u_i$.

$$v_k = \sum_{i,j} w_{kij} x_i u_j$$

   With the coding we use, it is easy to see that only one of the terms $x_i u_j$ is 1 with any given input and state. By taking the activation function to be the logistic sigmoid $1/(1+e^{-v_k})$, the ouput is close to 1 if $v_k \gg 0$ and close to 0 if $v_k \ll 0$. If for instance the current state is $(x_1, x_2) = (1, 0)$ (corresponding to state A) and the input is $(u_1, u_2) = (0, 1)$ (corresponding to input one), the output should be $(0, 1)$ (state B), i.e., $v_1 \ll 0$ and $v_2 \gg 0$. This means that $w_{112} \ll 0$ and $w_{212} \gg 0$. Similar considerations give $w_{111} \gg 0$, $w_{211} \ll 0$, $w_{121} \ll 0$, $w_{221} \gg 0$, $w_{122} \gg 0$ and $w_{222} \ll 0$.

   As long as the activation function is differentiable, it is possible to use for instance RTRL or BPTT to learn the weights from given examples.