Manual: 7.2.3. Recurrent Networks

The basic idea of a recurrent neural network is to make the future dependent upon the past by a similar form of function like the perceptron model. So, for instance, a very simple recurrent model could be $x(t+1) = a(W \cdot x(t) + b)$. Note very carefully three important features of this equation in contrast to the perceptron discussed before: (1) Both input and output are no longer vectors of values but rather vectors of functions that depend on time, (2) there is no separate input and output but rather the input and output are the same entity at two different times and (3) as this is a time-dependent recurrence relation, we need an initial condition for evaluation.

The network we are going to employ uses the concept of a reservoir, which is essentially just a set of nodes that are connected to each other in some way. The connection between nodes is expressed by a weight matrix W that is initialized in some fashion. Generally it is initialized randomly but substantial gain can be got when it is initialized with some structure. At present, it is a black art to determine what structure this should be as it definitely depends upon the problem to be solved. The current state of each node in the reservoir is stored in a vector $x(t)$ that depends on time t.

An input signal is given to the network $u(t)$ that also depends upon time. This is the actual time-series measured in reality that we wish to predict. The input process is governed by an input weight matrix W^{in} that provides the input vector values to any desired neurons in the reservoir. The output of the reservoir is given as a vector $y(t)$. The system state then evolves over time according to $x(t+1) = f(W \cdot x(t) + W^{in} \cdot u(t+1) + W^{fb} \cdot y(t))$ where W^{fb} is a feedback matrix, which is optional for cases in which we want to include the feedback of output back into the system and $f(...)$ is a sigmoidal function, usually $tanh(...)$.

The output $y(t)$ is computed from the extended system state $z(t) = [x(t); u(t)]$ by using $y(t) = g(W^{out} \cdot z(t))$ where W^{out} is an output weight matrix and $g(...)$ is a sigmoidal activation function, e.g. $tanh(...)$.

The input and feedback matrices are part of the problem specification and must therefore be provided by the user. The internal weight matrix is initialized in some way and then remains untouched. If the matrix W satisfies some complex conditions, then the network has the echo state property, which means that the prediction is eventually independent of the initial condition. This is crucial in that it does not matter at which time we begin modeling. Such networks are theoretically capable of representing any function (with some technical conditions) arbitrarily well if correctly set up.