1 Solving for the Weights

Here's my idea for solving for the weights in the type of neural nets where there is no hidden layer. First for some notation. Let's work with a two neuron network, which I will generalize later. Here is the dynamical system:

$$\begin{aligned} x_{1,t+1} &= \psi(w_{11}x_{1,t} + w_{12}x_{2,t}) & (1.1)\\ x_{2,t+1} &= \psi(w_{21}x_{1,t} + w_{22}x_{2,t}) & (1.2)\end{aligned}$$

where $\psi(u) = \tanh u$. Let the time series data be $\{d_0, d_1, \ldots\}$. I'll use the notation

$$d_t^k = (d_t, d_{t+1}, \ldots, d_{t+k-1})$$

for vectors of length k of the data. I'll also use the notation $\boldsymbol{w}_i = (w_{i1}, w_{i2}, \ldots, w_{in})$ for the rows of the matrix of weights, and $\boldsymbol{x}_t = (x_{1t}, x_{2t}, \ldots, x_{nt})$ for the vector of outputs at time t.

When training the net, we set $\boldsymbol{x}_0 = d_0^n$ and then try to find weights so that $|d_t^n - \boldsymbol{x}_t|$ are as close to 0 as possible for $t = 1, 2, \ldots, m$, where m is the length of the data set. Look at what this means for the $n = 2$ net when $m = 2$.

$$\begin{aligned} d_1 &\approx \psi(\boldsymbol{w}_1 \cdot d_0^2)\\ d_2 &\approx \psi(\boldsymbol{w}_2 \cdot d_0^2) \end{aligned} \qquad (1.3)$$

$$\begin{aligned} d_2 &\approx \psi(\boldsymbol{w}_1 \cdot \boldsymbol{x}_1)\\ d_3 &\approx \psi(\boldsymbol{w}_2 \cdot \boldsymbol{x}_1) \end{aligned} \qquad (1.4)$$

Now suppose that I can find weights so that equations (1.3) hold with equality. Then in equations (1.4) the value of $\boldsymbol{x}_1$ is d_1^2, and so we would really like to solve the following system

$$\begin{aligned} \psi(\boldsymbol{w}_1 \cdot d_0^2) &= d_1 & (1.5)\\ \psi(\boldsymbol{w}_2 \cdot d_0^2) &= d_2 & (1.6)\\ \psi(\boldsymbol{w}_1 \cdot d_1^2) &= d_2 & (1.7)\\ \psi(\boldsymbol{w}_2 \cdot d_1^2) &= d_3 & (1.8)\end{aligned}$$

Notice that equations (1.5) and (1.7) are two equations in two unknowns, w_{11} and w_{12}. Similarly, equations (1.6) and (1.8) are two equations in two unknowns, w_{21} and w_{22}. Also, unless the data are in $[-1, 1]$, there is no solution to these equations, so assume that the data are scaled to $|d_t| \leq 1$. More on scaling later. Now for the solution of equations (1.5) — (1.8). The vectors $\boldsymbol{w}_1$ and $\boldsymbol{w}_2$ satisfy the following *linear* system of equations,

$$\begin{bmatrix} d_0 & d_1 \\ d_1 & d_2 \end{bmatrix} \begin{bmatrix} w_{11} \\ w_{12} \end{bmatrix} = \begin{bmatrix} \psi^{-1}(d_1) \\ \psi^{-1}(d_2) \end{bmatrix} \qquad (1.9)$$

$$\begin{bmatrix} d_0 & d_1 \\ d_1 & d_2 \end{bmatrix} \begin{bmatrix} w_{21} \\ w_{22} \end{bmatrix} = \begin{bmatrix} \psi^{-1}(d_2) \\ \psi^{-1}(d_3) \end{bmatrix} \tag{1.10}$$

As long as $d_0 d_2 - d_1^2 \neq 0$, these equations have a unique solution. This is the generic case. From a numerical point of view, when $d_0 d_2 - d_1^2 \neq 0 \approx 0$, it is best to use the singular value decomposition method to solve for $\boldsymbol{w}_1$ and $\boldsymbol{w}_2$. There are routines in Press et al, *Numerical Recipes in C, 2nd Edition*, chapter 2.6.

For general n, we have n^2 weights to solve for. For $m = 2n$, we can proceed as above and fit the net *perfectly* to the first $2n$ data points. For simplicity of notation, let M be the matrix of data,

$$M = \begin{bmatrix} d_0 & d_1 & \cdots & d_n \\ d_1 & d_2 & \cdots & d_{n+1} \\ \vdots & \vdots & \ddots & \vdots \\ d_{n-1} & d_n & \cdots & d_{2n-1} \end{bmatrix}$$

and let $y_t = \psi^{-1}(d_t)$. Then

$$M\boldsymbol{w}_i = y_{i+1}^n \qquad i = 1, 2, \ldots, n \tag{1.11}$$

is the solution for the vectors $\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_n$. When we meet, I'll tell you about my ideas for when you want to train the net on $m > 2n$ data points.

2 Lyapunov Exponents

With these nets, you can compute the Lyapunov spectrum with Jacobian methods. The Jacobian of the net is

$$J_t = \begin{bmatrix} \psi'(\boldsymbol{w}_1 \cdot d_t^n)\boldsymbol{w}_1 \\ \psi'(\boldsymbol{w}_2 \cdot d_t^n)\boldsymbol{w}_2 \\ \vdots \\ \psi'(\boldsymbol{w}_n \cdot d_t^n)\boldsymbol{w}_n \end{bmatrix} \tag{2.12}$$

and the LEs can be calculated from the product

$$J_{n-1} J_{n-2} \cdots J_1 J_0 \quad .$$

3 Scaling

Here's where it gets interesting. Suppose that the unscaled data is $z_0, z_1, \ldots z_T$. How best to scale the data? You could use affine scaling, $d_t = s z_t$, so that $|d_t| < 1$ for all $0 \leq t \leq T$. Note that for very small s, the values of d_t would be very close to 0, and the net would operate in its (nearly) linear range. So, even if the data is from a chaotic source, the net would not produce chaotic output. This is the issue that Clint raised last week. On the other hand if s is large enough so that one or more of the scaled data, d_t, are close to 1, then $|\psi^{-1}(d_t)|$ will be very large, and the corresponding weights will also be large. That will push the neurons towards saturation.

One experiment to try is to solve for the weights as above and then compute the LEs for various scalings. The results might be very illuminating. It would also be interesting to produce the time series from such nets and see what happens to them. As well as calculate the LEs along the produced time series.