Training recurrent neural networks with backpropagation through time

James M. Murray
Center for Theoretical Neuroscience, Columbia University

March 5, 2018

Consider an RNN in which a time-dependent input vector $\mathbf{x}(t)$ provides input to a recurrently connected hidden layer described by activity vector $\mathbf{h}(t)$, and this activity is read out to form a time-dependent output $\mathbf{y}(t)$. Such a network, illustrated in Figure 1, is defined by the following equations:

$$\begin{aligned} h_i(t+1) &= h_i(t) + \frac{1}{\tau}\left[-h_i(t) + \phi\left(\sum_{j=1}^{N} W_{ij} h_j(t) + \sum_{j=1}^{N_x} W_{ij}^{\text{in}} x_j(t+1)\right)\right], \\ y_i(t) &= \sum_{j=1}^{N} W_{ij}^{\text{out}} h_j(t). \end{aligned} \tag{1}$$

For concreteness, we take the nonlinear function appearing in (1) to be $\phi(\cdot) = \tanh(\cdot)$. The goal is to train this network to produce a target output function $\mathbf{y}^*(t)$ given a specified input function $\mathbf{x}(t)$. One can then define the error as the difference between the target output and the actual output, and the loss function as the squared error integrated over time:

$$\begin{aligned} \varepsilon_i(t) &= y_i^*(t) - y_i(t), \\ L &= \frac{1}{2T}\sum_{t=1}^{T}\sum_{i=1}^{N_y}\left[\varepsilon_i(t)\right]^2. \end{aligned} \tag{2}$$

The goal of producing the target output function $\mathbf{y}^*(t)$ is thus equivalent to minimizing this loss function.

We now derive the learning rules for backpropagation through time (BPTT) [2] in for the RNN described

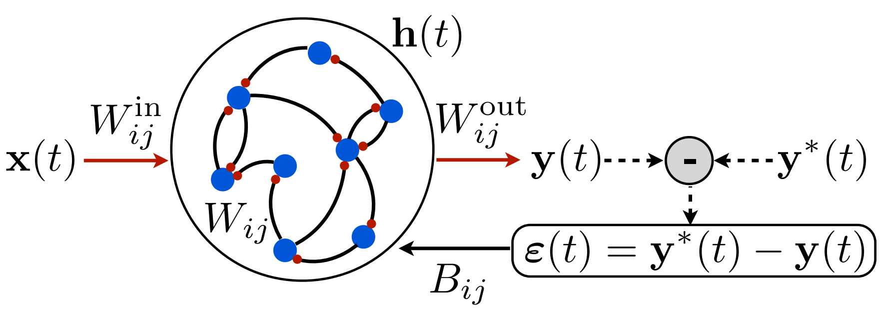

Figure 1: Schematic illustration of a recurrent neural network. In the case of BPTT, the error is projected back into the network for learning with weights $B_{ij} = W_{ij}^{\text{out}}$.

by (1). The derivation here follows Ref. [1]. Consider the following Lagrangian function:

$$\begin{aligned}\mathcal{L}\left[\vec{h},\vec{z},\mathbf{W},\mathbf{W}^{\text{in}},\mathbf{W}^{\text{out}},t\right] = \sum_i z_i(t)\left\{h_i(t)-h_i(t-1)+\frac{1}{\tau}\left[h_i(t-1)-\phi\left(\left[\mathbf{W}\vec{h}(t-1)+\mathbf{W}^{\text{in}}\vec{x}(t)\right]_i\right)\right]\right\} \\ +\frac{1}{2}\sum_i\left(y_i^*(t)-\left[\mathbf{W}^{\text{out}}\vec{h}(t)\right]_i\right)^2.\end{aligned} \tag{3}$$

The second line is the cost function that is to be minimized, while the first line uses the Lagrange multiplier $\vec{z}(t)$ to enforce the constraint that the dynamics of the RNN should follow (1). From (3) we can also define the following action:

$$S\left[\vec{h},\vec{z},\mathbf{W},\mathbf{W}^{\text{in}},\mathbf{W}^{\text{out}}\right] = \frac{1}{T}\sum_{t=1}^{T}\mathcal{L}\left[\vec{h},\vec{z},\mathbf{W},\mathbf{W}^{\text{in}},\mathbf{W}^{\text{out}},t\right]. \tag{4}$$

We now proceed by minimizing (4) with respect to each of its arguments. First, taking $\partial S/\partial z_i(t)$ just gives the dynamical equation (1). Next, we set $\partial S/\partial h_i(t)=0$, which yields

$$z_i(t) = \left(1-\frac{1}{\tau}\right)z_i(t+1)+\frac{1}{\tau}\left\{\sum_j z_j(t+1)\phi'\left(\left[\mathbf{W}\vec{h}(t)+\mathbf{W}^{\text{in}}\vec{x}(t+1)\right]_j\right)W_{ji}+\left[(W^{\text{out}})^T\boldsymbol{\varepsilon}(t)\right]_i\right\}, \tag{5}$$

which applies at timesteps $t=1,\ldots,T-1$. To obtain the value at the final timestep, we take $\partial S/\partial z_i(T)$, which leads to

$$z_i(T) = \left[(W^{\text{out}})^T\boldsymbol{\varepsilon}(T)\right]_i. \tag{6}$$

Finally, taking the derivative with respect to the weights leads to the following:

$$\begin{aligned}\frac{\partial S}{\partial W_{ij}} &= -\frac{1}{T\tau}\sum_{t=1}^{T}z_i(t)\phi'\left(\left[\mathbf{W}\vec{h}(t-1)+\mathbf{W}^{\text{in}}\vec{x}(t)\right]_i\right)h_j(t-1)\\ \frac{\partial S}{\partial W^{\text{in}}_{ij}} &= -\frac{1}{T\tau}\sum_{t=1}^{T}z_i(t)\phi'\left(\left[\mathbf{W}\vec{h}(t-1)+\mathbf{W}^{\text{in}}\vec{x}(t)\right]_i\right)x_j(t)\\ \frac{\partial S}{\partial W^{\text{out}}_{ij}} &= -\frac{1}{T}\sum_{t=1}^{T}\varepsilon_i(t)h_j(t).\end{aligned} \tag{7}$$

Rather than setting these derivatives equal to zero, which may lead to an undesired solution that corresponds to a maximum or saddle point of the action, we use the gradients in (7) to perform gradient descent, reducing the error in an iterative fashion:

$$\begin{aligned}\Delta W_{ij} &= \frac{\eta_2}{T\tau}\sum_{t=1}^{T}z_i(t)\phi'\left(\left[\mathbf{W}\vec{h}(t-1)+\mathbf{W}^{\text{in}}\vec{x}(t)\right]_i\right)h_j(t-1)\\ \Delta W^{\text{in}}_{ij} &= \frac{\eta_3}{T\tau}\sum_{t=1}^{T}z_i(t)\phi'\left(\left[\mathbf{W}\vec{h}(t-1)+\mathbf{W}^{\text{in}}\vec{x}(t)\right]_i\right)x_j(t)\\ \Delta W^{\text{out}}_{ij} &= \frac{\eta_1}{T}\sum_{t=1}^{T}\varepsilon_i(t)h_j(t),\end{aligned} \tag{8}$$

where η_i are learning rates.

The BPTT algorithm then proceeds in three steps. First the dynamical equations (1) for $\vec{h}(t)$ are integrated forward in time, beginning with the initial condition $\vec{h}(0)$. Second, the auxiliary variable $\vec{z}(t)$ is integrated *backwards* in time using (5), using with the $\vec{h}(t)$ saved from the forward pass and the boundary condition $\vec{z}(T)$ from (6). Third, the weights are updated according to (8), using $\vec{h}(t)$ and $\vec{z}(t)$ saved from the preceding two steps.

References

[1] Yann Lecun, *A theoretical framework for back-propagation*, Proceedings of the 1988 Connectionist Models Summer School, CMU, Pittsburg, PA, Morgan Kaufmann, 1988.

[2] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams, *Learning internal representations by error propagation*, Tech. report, California Univ San Diego La Jolla Inst for Cognitive Science, 1985.