
April 29, 2009

# Hopfield Network Learning Using Deterministic Latent Variables

*Tristan Fletcher*

www.cs.ucl.ac.uk/staff/T.Fletcher/

# 1 Introduction

This document explains how to use Barber's Deterministic Latent Tables [1] & [2] to learn the parameters in a Hopfield Network. The document concludes with Matlab code showing an implementation of the learning method detailed here.

In order to make the maths appear less cluttered, superscript indices such as the $t$ in $\mathbf{v}^t$ relate to time. Vectors are represented by bold font, and an element in a vector or array by subscript indices, so $\mathbf{v}_i^t$ is the $i^{th}$ element of vector $\mathbf{v}$ at time $t$.

With the aim of making the resultant Matlab implementation as transparent as possible, where horizontal space permits, some of the equations that are outlined are shown in both matrix form ($\mathbf{A}$) and element by element form ($A_{ij}$).

# 2 Deterministic Latent Tables

Deterministic Latent Tables are similar to traditional Hidden Markov Models except that the Conditional Probability Table (CPT) for the latent state transitions are deterministic instead of stochastic. This deterministic latent CPT is characterised as follows:

$$p(\mathbf{h}^{t+1}|\mathbf{v}^{t+1}, \mathbf{v}^t, \mathbf{h}^t) = \boldsymbol{\delta}(\mathbf{h}^{t+1} - \mathbf{f}(\mathbf{v}^{t+1}, \mathbf{v}^t, \mathbf{h}^t, \boldsymbol{\theta}_\mathbf{h})) \tag{2.1}$$

where $\boldsymbol{\delta}(x)$ represents the Dirac delta function for continuous hidden variables and the Kronecker delta for discrete hidden variables. The vector function $\mathbf{f}$ parameterises the CPT, itself having parameters $\boldsymbol{\theta}_\mathbf{h}$.

The log likelihood of a training sequence of $T$ observations $\mathbf{v}^{1:T}$ is:

$$L(\boldsymbol{\theta}_\mathbf{v}, \boldsymbol{\theta}_\mathbf{h}|\mathbf{v}^{1:T}) = \log p(\mathbf{v}^1|\boldsymbol{\theta}_\mathbf{v}) + \sum_{t=1}^{T} \log p(\mathbf{v}^{t+1}|\mathbf{v}^t, \mathbf{h}^t, \boldsymbol{\theta}_\mathbf{v}) \tag{2.2}$$

where the hidden unit values are calculated recursively using

$$\mathbf{h}^{t+1} = \mathbf{f}(\mathbf{v}^{t+1}, \mathbf{v}^t, \mathbf{h}^t, \boldsymbol{\theta}_\mathbf{h}) \tag{2.3}$$

The adjustable parameters of the hidden and visible CPTs are represented by $\boldsymbol{\theta}_\mathbf{h}$ and $\boldsymbol{\theta}_\mathbf{v}$ respectively. In order to maximise the log likelihood one needs to evaluate the derivatives with respect to these parameters. These can be calculated as follows:

$$\frac{dL}{d\boldsymbol{\theta}_\mathbf{v}} = \frac{\partial \log p(\mathbf{v}^1|\boldsymbol{\theta}_\mathbf{v})}{\partial \boldsymbol{\theta}_\mathbf{v}} + \sum_{t=1}^{T-1} \frac{\partial}{\partial \boldsymbol{\theta}_\mathbf{v}} \log p(\mathbf{v}^{t+1}|\mathbf{v}^t, \mathbf{h}^t, \boldsymbol{\theta}_\mathbf{v}) \tag{2.4}$$

$$\frac{dL}{d\boldsymbol{\theta}_\mathbf{h}} = \sum_{t=1}^{T-1} \frac{\partial}{\partial \mathbf{h}^t} \log p(\mathbf{v}^{t+1}|\mathbf{v}^t, \mathbf{h}^t, \boldsymbol{\theta}_\mathbf{v}) \frac{d\mathbf{h}^t}{d\boldsymbol{\theta}_\mathbf{h}} \tag{2.5}$$

$$\frac{d\mathbf{h}^t}{d\boldsymbol{\theta}_\mathbf{h}} = \frac{\partial \mathbf{f}^t}{\partial \boldsymbol{\theta}_\mathbf{h}} + \frac{\partial \mathbf{f}^t}{d\mathbf{h}^{t-1}} \frac{d\mathbf{h}^{t-1}}{d\boldsymbol{\theta}_\mathbf{h}} \tag{2.6}$$

where $\mathbf{f}^t = \mathbf{f}(\mathbf{v}^{t+1}, \mathbf{v}^t, \mathbf{h}^t, \boldsymbol{\theta}_\mathbf{h})$. Hence the derivatives can be calculated by deterministic propagation of errors.

# 3 Learning in a Hopfield Network

## 3.1 Model Specification

In a Hopfield Network [3], the probability of the next observation vector $\mathbf{v}^{t+1}$, given the previous one $\mathbf{v}^t$ and latent state $\mathbf{h}^t$ is described as follows:

$$p(\mathbf{v}^{t+1}|\mathbf{v}^t, \mathbf{h}^t) = \prod_{i=1}^{V} \sigma\left[(2v_i^{t+1} - 1)a_i^t\right] \tag{3.1}$$

where $\sigma(x) = 1/(1 + e^{-x})$, $V$ is the dimensionality of the observation vector $\mathbf{v}$, $H$ is the number of hidden neurons/dimensionality of $\mathbf{h}$ and:

$$\begin{aligned} \mathbf{a}^t &= \mathbf{W}\mathbf{v}^t + \mathbf{A}\mathbf{h}^t \\ a_i^t &= \sum_{j=1}^{V} W_{ij} v_j^t + \sum_{j=1}^{H} A_{ij} h_j^t \end{aligned} \tag{3.2}$$

and the latent state progresses recursively:

$$\begin{aligned} \mathbf{h}^{t+1} &= 2\sigma\left[\mathbf{B}\mathbf{h}^t + \mathbf{C}\mathbf{v}^t\right] - 1 \\ h_i^{t+1} &= 2\sigma\left[\sum_{j=1}^{H} B_{ij} h_j^t + \sum_{j=1}^{V} C_{ij} v_j^t\right] - 1 \end{aligned} \tag{3.3}$$

Note that (3.3) is equivalent to (2.3).

## 3.2 Parameter Learning

The log likelihood of a sequence of $T$ observations of $\mathbf{v}^t$ can then be expressed:

$$\begin{aligned} L &= \sum_{t=1}^{T-1} \sum_{i=1}^{V} \log \sigma\left[(2v_i^{t+1} - 1)a_i^t\right] + \dots \\ &= \sum_{t=1}^{T-1} \sum_{i=1}^{V} \log \sigma\left[(2v_i^{t+1} - 1)\left(\sum_{j=1}^{V} W_{ij} v_j^t + \sum_{j=1}^{H} A_{ij} h_j^t\right)\right] + \dots \end{aligned} \tag{3.4}$$

Differentiating the log likelihood shown in (2.4) with respect to the observable parameters $\boldsymbol{\theta}_v \in \{\mathbf{W}, \mathbf{A}\}$:

$$\begin{aligned} \frac{dL}{dW_{ij}} &= \sum_{t=1}^{T-1} \left(1 - \sigma\left[(2v_i^{t+1} - 1)a_i^t\right]\right)(2v_i^{t+1} - 1)v_j^t \\ \frac{dL}{dA_{ij}} &= \sum_{t=1}^{T-1} \left(1 - \sigma\left[(2v_i^{t+1} - 1)a_i^t\right]\right)(2v_i^{t+1} - 1)h_j^t \end{aligned} \tag{3.5}$$

Differentiating the log likelihood with respect to the latent parameters $\boldsymbol{\theta}_h \in \{\mathbf{C}, \mathbf{B}\}$ one first needs to calculate:

$$\begin{aligned}
\frac{\partial L}{\partial \mathbf{h}^t} &= \sum_{t=1}^{T-1} \mathbf{A} \left(1 - \sigma\left[(2\mathbf{v}^{t+1} - 1)\mathbf{a}^t\right]\right)(2\mathbf{v}^{t+1} - 1) \\
\frac{\partial L}{\partial h_j^t} &= \sum_{t=1}^{T-1} \sum_{i=1}^{V} A_{ij} \left(1 - \sigma\left[(2v_i^{t+1} - 1)a_i^t\right]\right)(2v_i^{t+1} - 1)
\end{aligned} \tag{3.6}$$

$\frac{d\mathbf{h}^t}{d\boldsymbol{\theta}_h}$ then needs to be calculated for each parameter:

$$\begin{aligned}
\frac{d\mathbf{h}^t}{d\mathbf{C}} &= \frac{\partial \mathbf{f}^t}{\partial \mathbf{C}} + \frac{\partial \mathbf{f}^t}{\partial \mathbf{h}^{t-1}} \frac{d\mathbf{h}^{t-1}}{d\mathbf{C}} \\
&= 2\boldsymbol{\sigma}^{t-1}(1 - \boldsymbol{\sigma}^{t-1})\mathbf{v}^{t-1} \\
\frac{dh_i^t}{dC_{ij}} &= 2\sigma\left[\sum_{j=1}^{H} B_{ij} h_j^{t-1} + \sum_{j=1}^{V} C_{ij} v_j^{t-1}\right]\left(1 - \sigma\left[\sum_{j=1}^{H} B_{ij} h_j^{t-1} + \sum_{j=1}^{V} C_{ij} v_j^{t-1}\right]\right) v_j^{t-1}
\end{aligned} \tag{3.7}$$

and

$$\begin{aligned}
\frac{d\mathbf{h}^t}{d\mathbf{B}} &= \frac{\partial \mathbf{f}^t}{\partial \mathbf{B}} + \frac{\partial \mathbf{f}^t}{\partial \mathbf{h}^{t-1}} \frac{d\mathbf{h}^{t-1}}{d\mathbf{B}} \\
&= 2\boldsymbol{\sigma}^{t-1}(1 - \boldsymbol{\sigma}^{t-1})\left(\mathbf{h}^{t-1} + \mathbf{B}\frac{dh^{t-1}}{d\mathbf{B}}\right)
\end{aligned} \tag{3.8}$$

where $\boldsymbol{\sigma}^t = \sigma\left[\mathbf{B}\mathbf{h}^t + \mathbf{C}\mathbf{v}^t\right]$.

Substituting (2.6) and (2.7) into (1.5):

$$\frac{dL}{d\mathbf{C}} = \sum_{t=1}^{T-1} \mathbf{A} \left(1 - \sigma\left[(2\mathbf{v}^{t+1} - 1)\mathbf{a}^t\right]\right)(2\mathbf{v}^{t+1} - 1)(2\boldsymbol{\sigma}^{t-1}(1 - \boldsymbol{\sigma}^{t-1}))\mathbf{v}^{t-1} \tag{3.9}$$

And substituting (2.6) and (2.8) into (1.5):

$$\frac{dL}{d\mathbf{B}} = \sum_{t=1}^{T-1} \mathbf{A} \left(1 - \sigma\left[(2\mathbf{v}^{t+1} - 1)\mathbf{a}^t\right]\right)(2\mathbf{v}^{t+1} - 1)2\boldsymbol{\sigma}^{t-1}(1 - \boldsymbol{\sigma}^{t-1})\left(\mathbf{h}^{t-1} + \mathbf{B}\frac{dh^{t-1}}{d\mathbf{B}}\right) \tag{3.10}$$

Parameter learning then proceeds by iterating for each parameter:

$$\boldsymbol{\theta}^{new} = \boldsymbol{\theta} + \eta \frac{dL}{d\boldsymbol{\theta}} \tag{3.11}$$

# 4 Matlab Code

## 4.1 Initialise a Network

```
% Set Simulation Dimensions
T = 10;             % Number of time steps
V = 7;              % Dimension of visible vector
H = 4;              % Number of Neurons (latent states)

% Create training sequence
v = randn(V,T)>0;

% Initialise A,B and C with random values
A = 0.1*randn(V,H);
B = randn(H,H);
C = randn(H,V);

% Initialise weights to zero
W = zeros(V,V);

% Initialise latent states
h(:,1)=2*(sprandn(H,1,0.1)>0)-1;
for t = 1:T-1
    h(:,t+1)= 2*sig(B*h(:,t)+C*v(:,t))-1;
end
```

## 4.2 Learn Parameters

```
% Learn parameters A, B, C and W
for loop=1:epochs

    LL=0;   % Initialise Log Likelihood to zero

    % Update latent states
    for t = 1:T-1
        h(:,t+1)= 2*sig(B*h(:,t)+C*v(:,t))-1;
    end

    % Initalise parameter derivatives to zero
    dLdA = zeros(V,H);
    dLdB = zeros(H,H);
    dLdC = zeros(H,V);
    dLdW = zeros(V,V);
    dhmdB_old=zeros(H,H,H);

    % Sum over t
    for t = 1:T-1

        at = W*v(:,t) + A*h(:,t);
        dLdW = dLdW + (1-sig((2*v(:,t+1)-1).*at)).*(2*v(:,t+1)-1)*v(:,t)';
        dLdA = dLdA + (1-sig((2*v(:,t+1)-1).*at)).*(2*v(:,t+1)-1)*h(:,t)';

        if t>1

          sm = sig(B*h(:,t-1)+C*v(:,t-1));
          dLdC = dLdC + (A'*((1-sig((2*v(:,t+1)-1).*at)).*(2*v(:,t+1)-1))...
              .*(2*sm.*(1-sm)))*v(:,t-1)';
          sA=A'*((1-sig((2*v(:,t+1)-1).*at)).*(2*v(:,t+1)-1));

            for i=1:H

                for j=1:H

                    for m=1:H

                        sumr=0;

                        for r=1:H
                            sumr=sumr+B(m,r)*dhmdB_old(r,i,j);
                        end %r

                        if m==i
                            sumr=sumr+h(j,t-1);
                        end %if

                        dhmdB(m,i,j) = 2*sm(m).*(1-sm(m))*sumr;

                    end  %m

                    dLdB(i,j)=dLdB(i,j)+sA(m)*dhmdB(m,i,j);

                end %j
```

```
            end %i

            dhmdB_old=dhmdB;

        end %if

        LL = LL + sum(log(sig((2*v(:,t+1)-1).*at)));

    end %t

  % Store Log Likelihood
  L(loop)=LL;

  % Update parameters
  A = A + eta*dLdA;
  B = B + eta*dLdB;
  C = C + eta*dLdC;
  W = W + eta*dLdW;

end %loop

% Sigma function
function s = sig(x)
s = 1./(1+exp(-x));
```

### 4.3 Reconstruct Network

```
initnoise = 0.3;    % Reconstruction noise

% Flip some bits randomly at t=1
vr(:,1)=v(:,1);
toflip = find(rand(V,1)<initnoise);
vr(toflip,1) = 1-vr(toflip,1);
hr(:,1)=h(:,1);

% Reconstruct network from learned parameters
for t = 1:T-1
    at=W*vr(:,t) + A*hr(:,t);
    vr(:,t+1) = (at>0);
    hr(:,t+1)= 2*sig(B*hr(:,t)+C*vr(:,t))-1;
end

% Plot initial networks, reconstructed network, W and A
subplot(2,2,1); imagesc(v); colormap('gray'); title('Initial Network')
subplot(2,2,2); imagesc(vr); colormap('gray'); title('Reconstructed Network')
subplot(2,2,3); imagesc(W); colormap('gray'); title('W')
subplot(2,2,4); imagesc(A); colormap('gray'); title('A')
```

## References


[1] D. Barber, "Dynamic Bayesian Networks with Deterministic Tables," in *Advances in Neural Information Processing Systems (NIPS)*, 2003.

[2] ——, "Learning in Spiking Neural Assemblies," in *Advances in Neural Information Processing Systems (NIPS)*, 2003.

[3] R. P. John Hertz and A. Krogh, *Introduction to the Theory of Neural Computation*. Perseus Publishing, 1991.