NETWORKS AND COMPUTATIONS OF ARTIFICIAL NEURONS

Yuan Chen and Jonathan Mei

18-859 Lecture Notes: 11/7/2014 - 11/14/2014

1. HOPFIELD NETWORK

A Hopfield network is a completely connected directed graph of I nodes ("neurons"), i.e., a graph of I nodes, denoted $\{1, 2, \ldots, I\}$, in which there is a directed edge from each node to every other node (there are no self directed loops). Each node i has an associated output value x_i. For simplicity, we consider $x_i \in \{-1, 1\}$. From a biological perspective, such an binary assignment of node values corresponds to a neuron not firing ($x_i = -1$) and a neuron firing at maximum rate ($x_i = 1$), respectively [1]. Finally, there is a weight assigned to each edge: w_{ij} is the weight of the directed edge from node j to node i. The value of every node represents the state or output of the Hopfield network (i.e., the network of the state is a vector $x \in \{-1, 1\}^I$.

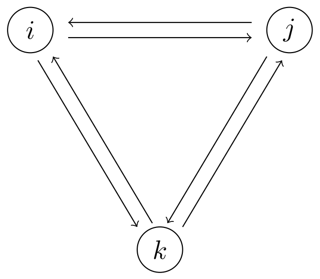

Fig. 1. A Hopfield network with $I = 3$ nodes

Each node in the Hopfield network has an update rule: the node changes its value over time based on the values of the other nodes in the network and the weights of the directed edges to that node. Let t be a time index. Define for every node i an activation level

$$a_i(t) = \sum_{j \neq i} w_{ij} x_j(t). \tag{1}$$

The node updates its value in the next interval according to some function$f(\dot{})$ of the activation $a_i(t)$.

$$x_i(t) = f(a_i(t)). \tag{2}$$

For the specific nodes we consider, the function f is a function of the form $f : \mathbb{R} \to \{-1, 1\}$ (the domain of f is $\mathbb{R}$ since we have not specified the particular weights of the network and have assumed that the weights can be any real number).

Node values can be updated either synchronously or asynchronously. In synchronous update, every node updates its value in each time step using the values of the all the nodes in the previous time step. That is, in synchronous update,

$$x(t+1) = F\left(Wx(t)\right), \tag{3}$$

where $x(t) \in \{-1, 1\}^I$ is the vector of all node values at time t, W is a matrix consisting of the edge weights, and F is a function that performs the element wise mapping of f on every element of the vector $Wx(t)$. In asynchronous update, only one node updates its value at each time step. That is, we only update the activation and value of a single node i at a particular time step t (according to equations (1) and (**??**), while maintaining the node values at all other nodes. Then, at the next time step, we update the activation and node value of node $i+1$ (if at time t we update node I, then at time $t+1$ we update node 1). Note that the time step in asynchronous update is not necessary of the same "duration" as the time step in synchronous update. For practical purposes, we can consider the iteration through the nodes in asynchronous update to take place over auxiliary time steps, in which I auxiliary time steps is equal to 1 time step in t for synchronous update. The actual duration of the time step does not affect the behavior of the model.

Equations (1) and (2) describe the dynamics of Hopfield network given the edge weight matrix W and function f. We consider an input to the system to be an initial setting of $x(0)$ (i.e., an initial setting of all node values), and we consider the output of the system to be the node values (over time) at all of the nodes. Suppose there are locally stable points $x^{(1)}, \ldots, x^{(N)}$ in the system. These are points such that if we give system input $x^i(s)$, the i^{th} locally stable point, then the output of the system $x(t) = x^{(i)}$ for all time steps t. We consider these stable points to be a particular "memory" associated with the Hopfield network [1]. Further, we seek for these stable points to be attractive. That is if we give the system initial state an initial state that is "close" to a stable point $x^{(i)}$ (for our nodes, we consider a Hamming distance metric), then the system state $x(t)$ will tend toward $x^{(i)}$ as t grows.

The stable points of a particular Hopfield network depend on the function f and the weight matrix W. We consider f to be a property of the nodes that is chosen separately from deciding the stable points. The function f can be chosen as a threshold function for our network. Thus, in order to encode stable points into a network, we must assign specific weights to the edges of the network.

2. CONVERGENCE IN HOPFIELD NETWORKS

Convergence (and existence of stable points) in Hopfield networks also depends on the update scheme (i.e. synchronous versus asynchronous update). Depending on the update scheme, certain inputs may never converge. For example, let's consider a Hopfield network with $I = 2$ nodes and the weight matrix

$$W = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \tag{4}$$

and the f function

$$f(a_i(t)) = a_i(t). \tag{5}$$

We first consider the synchronous update scheme with input $x(0) = \begin{bmatrix} -1 \\ 1 \end{bmatrix}$. Following the synchronous update rule given in equation (3), we have that $x(1) = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$, $x(2) = \begin{bmatrix} -1 \\ 1 \end{bmatrix}$, and in general, $x(t+1) = -x(t)$. Thus with the synchronous update rule and the input $x(0)$, the Hopfield network will never converge to a stable point and oscillates between two states for all time. Next, we consider asynchronous update in which we first update $x_1(t)$. In the initial time step, we have $a_1(0) = 1$. So updating, $x_1(1) = a_1(0)$ and $x_2(1) = x_2(0)$. Thus, under asynchronous update, we have $x(1) = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$. Then in all subsequent update states, we have $x(t+1) = x(t)$ (for $t = 1, 2, \dots$). Under an asynchronous update scheme, the input $x(0)$ to the example Hopfield network converges to the point $\begin{bmatrix} 1 \\ 1 \end{bmatrix}$ (a stable point of the example Hopfield network).

In fact, asynchronous update with a hard threshold function f, i.e.,

$$f(a_i(t)) = \begin{cases} 1, & a_i(t) \geq 0 \\ -1, & a_i(t) < 0 \end{cases} \tag{6}$$

always converges to a stable point $x^{(i)}$ of the Hopfield network. To prove this, we define an energy function

$$E = -x^T W x. \tag{7}$$

Note that E has a finite lower bound. This is true because x can only take on a finite set of values, and, in turn, E can only take on a set of finite real (finite) values. Viewing E as a function of x, this means there exists a global minimum of E, and there may exist local minima of E. We define local minima of E as points x^* such that $E(x^*) < E(x)$ for all x of Hamming distance 1 from x^*. We claim that the local minima and the global minimum of E are stable points of the Hopfield network, and for any input $x(0)$,the state of the Hopfield network converges to one of these stable points.

To prove convergence, we consider E as a function of t (since x changes with t), and we define

$$\Delta E(t) = E(t+1) - E(t). \tag{8}$$

$\Delta E(t)$ describes the change in the energy function when we update $x(t)$. Equation (8) can be expressed as

$$\Delta E(t) = -\sum_{i=1}^{I}\sum_{j=1}^{I} w_{ji}\left(x_i(t+1)x_j(t+1) - x_i(t)x_j(t)\right). \tag{9}$$

Because the node values are updated asynchronously, only one node, say node k, can change its value from time t to time $t+1$. Moreover, since defined the Hopfield network to have no self loops, we have $w_{kk} = 0$. Thus, equation (9) can be simplified to

$$\Delta E(t) = -\sum_{i=1}^{I} w_{ki} x_i(t)\left(x_k(t+1) - x_k(t)\right). \tag{10}$$

Define

$$\Delta x_k = x_k(t+1) - x_k(t). \tag{11}$$

By construction, Δx_k can take on the values of 2 (if x_k changes from -1 to 1), -2 (if x_k changes from 1 to -1), or 0 (if x_k does not change). Substituting equation (11) into equation (9), we have

$$\Delta E(t) = -a_k(t)\Delta x_k, \tag{12}$$

where $a_k(t)$ is the activation at node k. In order for Δx_k to be positive, $x_k(t+1) = 1$, which by equation (6) means that $a_k(t) \geq 0$. Similarly, in order for Δx_k to be negative, $x_k(t+1) = -1$, which by equation (6) means that $a_k(t) < 0$. By this argument, we have that $\Delta E(t) \leq 0$.

To conclude the proof of convergence, $\Delta E(t) \leq 0$ means that at every update, x changes so that E is non-increasing. Because E is bounded below, x cannot be changing in each step in a way such that E is always decreasing. Thus, x tends toward the local minima and global minima of x. Since by definition, the minima (both local and global) are separated by Hamming distance greater than 1, the state of the Hopfield network cannot move from minima to minima in asynchronous update. Furthermore, since, by definition, the value of E at all x of Hamming distance 1 from a minima $x*$ is greater than $E(x*)$ and since ΔE is nonnegative, once the state of the system reaches $x*$ it remains at $x*$.

3. CAPACITY OF HOPFIELD NETWORKS

Here, we introduce Hebbian Learning Rule and analyze the capacity (in the traditional information theory sense) of a

Hopfield Network with I neurons and N stored memories under this rule. In this setting, we treat the inputs as message bits, and the attractor points as codewords. We see that to take the probability of error $P(E) \to 0$, we need the number of memories $N = o(I)$. This final result of the analysis suggests that there is a cost to having additional properties (e.g. distributed learning/update rules, associative memories, etc.) in storing memories in HN's.

3.1. Errors in Hopfield Networks

In order to analyze limits of storing information in HN's, we first need to consider the types of errors that can occur in HN's. Consider a Hopfield Network with I neurons and N memories $\{\mathbf{x}^{(n)}\}$ where $\mathbf{x} \in \mathbb{R}^I$. In trying to access a memory $\mathbf{x}^{(n)}$, there are several types of errors that may be observed,

1. Bit errors, in which the stable state $\mathbf{x}^*$ is slightly distorted version of $\mathbf{x}^{(n)}$, i.e. $0 < \|\mathbf{x}^* - \mathbf{x}^{(n)}\| < \epsilon$
2. Missing memories, in which $\mathbf{x}^{(n)}$ is not a stable point and there is no nearby point (for example in some radius $\|\mathbf{x} - \mathbf{x}^{(n)}\| \leq r$) that is stable
3. Small basin of attraction. Not a true "error" but is not desirable
4. Spurious additional unrelated state, in which an existing stable point $\mathbf{x}^*$ does not correspond to any desired memory, i.e. $\|\mathbf{x}^* - \mathbf{x}^{(n)}\| \geq \epsilon \; \forall n$
5. Spurious additional related state, in which there are stable points $\mathbf{x}^*$ corresponding to $\mathbf{x}^{(n)}$

3.2. Capacity of Linear Hebbian Hopfield Networks

What is the relationship between number of memories N and number of neurons I that is needed to achieve a given error probability ϵ? We consider a randomized experiment to answer this question, considering only error type (1) from above for a single input at a single node. Because we only consider this one type of error, the true probability of error is higher. Thus we are quantifying the capacity optimistically.

Suppose our memories with $\mathbf{x}_i^{(n)} \in \{-1, +1\}$ are generated as i.i.d. $\mathbf{x}^{(n)} \sim$Bernoulli(0.5). Assume a network of *linear* neurons with outputs

$$y_i = \sum_{j \neq i} W_{ij} x_j$$

Also assume the learning rule for the network is Hebbian, and we learn the weights as

$$W_{ij} = \sum_{n=1}^{N} x_i^{(n)} x_j^{(n)}$$

After this experiment, we have a learned network ready to accept inputs. we wish to find the probability that a learned memory has a bit error with respect to the true desired $\mathbf{x}^{(n)}$. Without loss of generality, consider the input $\mathbf{x}^{(1)}$.

The learned weights are then

$$W_{ij} = x_i^{(1)} x_j^{(1)} + \sum_{n=2}^{N} x_i^{(n)} x_j^{(n)}$$

The activity at node i is

$$\begin{aligned} y_i &= \sum_{i=\neq j} \left(x_i^{(1)} x_j^{(1)} x_j^{(1)} + \sum_{n=2}^{N} x_j^{(n)} x_j^{(n)} x_j^{(1)} \right) \\ &\approx (I-1) x_i^{(1)} + \mathcal{N}(0, (I-1)(N-1)) \end{aligned}$$

Then the probability for error is approximately

$$P(\text{Bit i flips in 1st update}) \approx \mathbf{\Phi}(-\frac{1}{\sqrt{N/I}})$$

We see that holding the rate $\log(N)/I$ constant, P(error) is not decreasing.

4. COMPUTATION OF A SINGLE LINEAR NEURON

In this section, we examine the computation being performed by a *single linear* neuron with output,

$$y(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} = \sum_i w_i x_i \tag{13}$$

We make a modification to the Hebbian learning rule to arrive at Oja's rule. We show that this particular neuron with Oja's rule [2] is in a way performing *Principal Component Analysis*.

4.1. Departing from Hebbian Learning

The first modification to the Hebbian learning rule is a normalization,

$$w_i' = \frac{w_i + \eta y(\mathbf{x}) x_i}{\|\mathbf{w} + \eta y(\mathbf{x}) \mathbf{x}\|} = D\left(w_i + \eta y(\mathbf{x}) x_i\right) \tag{14}$$

where $D = (\|\mathbf{w} + \eta y(\mathbf{x})\mathbf{x}\|)^{-1}$. Instead of studying this modified learning rule, we next find an approximation of this rule for small step size η. Expanding D through its Taylor series,

$$\begin{aligned} D &= \left(\sum_j \left(w_j^2 + 2\eta y(\mathbf{x}) x_j w_j + O(\eta^2) \right) \right)^{-1/2} \\ &\approx \left(\sum_j w_j^2 + 2\eta y(\mathbf{x}) \sum_j x_j w_j \right)^{-1/2} \\ &\approx (1 + 2\eta y^2(\mathbf{x}))^{-1/2} \approx 1 - \eta y^2(\mathbf{x}) \end{aligned}$$

where the penultimate step follows from the fact that $\|\mathbf{w}\| \approx 1$ from continually normalizing at each step in our rule and from our linear neuron output relation in 13. Returning to our normalized learning rule from 14,

$$\begin{aligned} w_i' &\approx (1-\eta y^2(\mathbf{x}))\,(w_i + \eta y(\mathbf{x}) x_i) \\ &= w_i + \eta y(\mathbf{x}) - \eta y^2(\mathbf{x}) w_i + O(\eta^2) \\ &\approx w_i + \eta y(\mathbf{x})(x_i - y(\mathbf{x}) w_i) \\ \Rightarrow \Delta\mathbf{w} &= \eta y(\mathbf{x})(\mathbf{x} - y(\mathbf{x})\mathbf{w}) \end{aligned} \tag{15}$$

This is Oja's rule (compare to Hebbian learning rule $\Delta\mathbf{w} = \eta y(\mathbf{x})\mathbf{x}$).

4.2. Behavior of Oja's Rule

Oja's rule computes a vector $\mathbf{w}$ that satisfies the following properties:

1. $\|\mathbf{w}\| \to 1$
2. $\mathbf{w}$ tends to an eigenvector v_1 of the covariance matrix $C = \mathbb{E}[\mathbf{x}\mathbf{x}^\top]$
3. The eigenvalue λ_1 corresponding to eigenvector v_1 is the *largest* eigenvalue of C. In other words, v_1 is the principal vector.

The full analysis of Oja's rule is difficult, so we will present an analysis *at equilibrium*.

4.2.1. Properties 1 & 2

Assuming we are at equilibrium $\mathbf{w}^*$,

$$\begin{aligned} &\frac{1}{\eta}\mathbb{E}[\Delta\mathbf{w}^*] = \mathbf{0} \\ \Rightarrow &\mathbf{0} = \mathbb{E}[y(\mathbf{x})(\mathbf{x} - y(\mathbf{x})\mathbf{w}^*)] = \mathbb{E}[\mathbf{x}\cdot\mathbf{x}^\top\mathbf{w}^* - \mathbf{w}^{*\top}\mathbf{x}\cdot\mathbf{x}^\top\mathbf{w}^*\cdot\mathbf{w}^*] \\ \Rightarrow &C\mathbf{w}^* - (\mathbf{w}^{*\top}C\mathbf{w}^*)\cdot\mathbf{w}^* = C\mathbf{w}^* - \lambda^*\mathbf{w}^* = \mathbf{0} \end{aligned}$$

where $\lambda^* = \mathbf{w}^{*\top}C\mathbf{w}^*$. Then we have shown property (2) holds at equilibrium, namely that $\mathbf{w}^*$ is an eigenvector of C, with a corresponding eigenvalue of λ^*. Then,

$$\begin{aligned} \lambda^* &= \mathbf{w}^\top C\mathbf{w}^* = \mathbf{w}^{*\top}\lambda^*\mathbf{w}^* = \lambda\|\mathbf{w}^*\|^2 \\ \Rightarrow &\|\mathbf{w}^*\| = 1 \end{aligned}$$

shows property (1) holds at equilibrium.

4.2.2. Property 3

Now we wish to show that the stable equilibrium point is the eigenvector v_1 with maximum eigenvalue λ_1. Let $\mathbf{v}$ be some other eigenvector with eigenvalue λ. While $\mathbf{v}$ is an equilibrium point, we will show it is unstable if $\lambda < \lambda_1$. Starting at $\mathbf{w} = \mathbf{v} + \boldsymbol{\epsilon}$ with a small deviation $\|\boldsymbol{\epsilon}\|$ from $\mathbf{v}$,

$$\begin{aligned} \frac{1}{\eta}\mathbb{E}[\Delta\boldsymbol{\epsilon}] = \frac{1}{\eta}\mathbb{E}[\Delta\mathbf{w}] &= C\mathbf{w} - (\mathbf{w}^\top C\mathbf{w})\mathbf{w} \\ &= C\mathbf{v} + C\boldsymbol{\epsilon} - (\mathbf{v}^\top C\mathbf{v} + \mathbf{v}^\top C\boldsymbol{\epsilon} + \boldsymbol{\epsilon}^\top C\mathbf{v} + \boldsymbol{\epsilon}C\boldsymbol{\epsilon})(\mathbf{v} + \boldsymbol{\epsilon}) \\ &\approx \lambda\mathbf{v} + C\boldsymbol{\epsilon} - (\lambda + 2\lambda\mathbf{v}^\top\boldsymbol{\epsilon})(\mathbf{v} + \boldsymbol{\epsilon}) \\ &= C\boldsymbol{\epsilon} - (2\lambda\mathbf{v}\mathbf{v}^\top\boldsymbol{\epsilon} + \lambda\boldsymbol{\epsilon} + 2\lambda\boldsymbol{\epsilon}\boldsymbol{\epsilon}^\top\mathbf{v}) \\ &\approx C\boldsymbol{\epsilon} - 2\lambda\mathbf{v}\mathbf{v}^\top\boldsymbol{\epsilon} - \lambda\boldsymbol{\epsilon} \end{aligned}$$

where we have discarded terms of $O(\|\boldsymbol{\epsilon}\|^2)$. Now,

$$\begin{aligned} \frac{1}{\eta}v_1^\top\mathbb{E}[\Delta\boldsymbol{\epsilon}] &\approx \mathbf{v}_1^\top C\boldsymbol{\epsilon} - 2\lambda\mathbf{v}_1^\top\mathbf{v}\mathbf{v}^\top - \lambda\mathbf{v}_1^\top\boldsymbol{\epsilon} \\ &= (\lambda_1 - \lambda)\mathbf{v}_1^\top\boldsymbol{\epsilon} \\ \Rightarrow \mathbf{v}_1^\top\mathbb{E}[\boldsymbol{\epsilon} + \Delta\boldsymbol{\epsilon}] &\approx (1 + \eta(\lambda_1 - \lambda))\mathbf{v}_1^\top\boldsymbol{\epsilon} \end{aligned}$$

since symmetric matrices yield orthogonal eigenvectors $\mathbf{v}_1^\top\mathbf{v} = 0$. Note that since $\eta > 0$ and $\lambda_1 - \lambda > 0$, the multiplicative term $1 + \eta(\lambda_1 - \lambda) > 1$. This implies that the magnitude of $\mathbb{E}[\boldsymbol{\epsilon}]$ in the direction of $\mathbf{v}_1$ is increasing. Thus, $\mathbf{v}_1$ is a stable point, and we have shown property (3).

5. REFERENCES

[1] J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities," *Proceedings of the National Academy of Sciences*, vol. 79, pp. 2554–2558, Apr. 1986.

[2] Erkki Oja, "Simplified neuron model as a principal component analyzer," *Journal of mathematical biology*, vol. 15, no. 3, pp. 267–273, 1982.