Motivation/Rationale

'Remembering' something in common parlance usually consists of associating something with a sensory cue. For example someone may say something like the name of a celebrity and we immediately recall a chain of events or some experience related to the celebrity we may have seen them on TV recently for example. Or we may see a picture of a place visited in our childhood and the image recalls memories of the time.

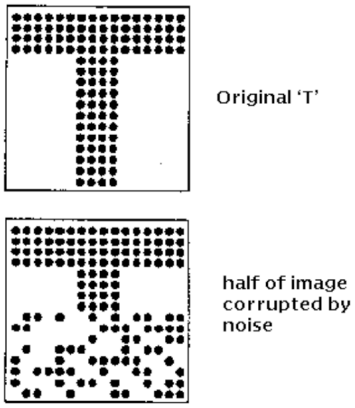

Fig. 1: Distorted pattern

The common paradigm here may be described as follows. There is some underlying collection of data which is ordered and interrelated in some way and which is stored in memory. The data may be thought of therefore as forming a stored pattern. In the recollection examples above it is the cluster of memories associated with the celebrity or the phase in childhood. In the case of character recognition it is the parts of the letter (pixels) whose arrangement is determined by an archetypal version of the letter. When part of the pattern of data is presented in the form of a sensory cue, the rest of the pattern memory is recalled or associated with it. Notice that it often doesn't matter which part of the pattern is used as the cue the whole pattern is always restored.

Syllabus

Unit 5 – Associative Models **[__ hrs.]**

5.1. Linear Associative Memory (LAM)
5.2. Hopfield Networks
5.3. Brain-State-in-a-Box (BSB)
5.4. Boltzmann Machines and Simulated Annealing
5.5. Bi-Directional Associative Memory (BAM)

ASSOCIATIVE MODELS

Associative models treat every input sample as a separate class. Therefore, these models create a one-to-one relationship between input vectors and output vectors. Various types of associative models exist. Some models return an output vector that has the same size as of input vector whereas some other models return output vector with different size than input vector.

There are two types of associative networks, hetero-associative (HA) and auto-associative (AA). Hetero-associative takes an input vector and outputs a completely *different* vector. For example, an HA-net could be used to take a sound file and output the text that it represents (or the closest learned text). Auto-associative networks take input and output the same vector. How is this useful? An AA-net could learn various vectors, then when a corrupted vector came in, it would be corrected by the net. This can be used in image-recognition of partial or corrupted ("noisy") images. It is auto-associative networks that this essay will focus on, or more specifically the Hopfield network.

Linear Associative Memory (LAM)

"The general idea is an old one, that any two cells or systems of cells that are repeatedly active at the same time will tend to become **'associated'**, so that activity in one facilitates activity in the other." (Hebb 1949, p. 70)

Hopfield Networks

A Hopfieldnet (Hopfield1982) is a net of such units subject to the asynchronous rule for updating one neuron at a time:

"Pick a unit **i** at random.
If $\Sigma\, w_{ij}\, s_j \geq \theta_i$, turn it on.
Otherwise turn it off."

Where, S is state of unit 'j' and 'θ_i' is threshold of unit 'i'.

Moreover, Hopfield assumes symmetric weights: $w_{ij} = w_{ji}$.

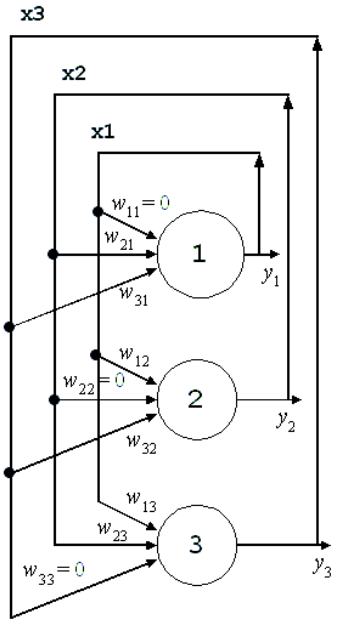

Fig. 2: Hopfield Network

The Learning Matrix

Generally considered to be fixed-weight models, they **don't learn**. However, one way to get the weights is through the supervised Hebbian outer-product summation. To be effective, the patterns should be reasonably **orthogonal**.

The Learning Matrix was an electronic device that would learn a desired output, given a certain input — it would do this by adjusting the weights within the matrix using a Hebbian learning rule, when given the input and output vectors. The reason this is important is because it introduces the matrix mathematic behind the Hopfield network. The learning matrix operation could be summarized mathematically as:

$$[Y] = [X][W]$$
$$[W] = \eta[X]^t[Y]$$

Where [X] is the input matrix (group of patterns to be learnt), [Y] is the output matrix, and [W] is the weights. Let's see a simple example. To keep the matrices small, consider that there are only two patterns with four bits.

$$[X] = \begin{bmatrix} 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 \end{bmatrix}$$
$$[Y] = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix}$$

To make calculations a lot simpler, we'll make η = 1. Now, to calculate the weights that will give us our output given the input:

$$[X]^t = \begin{bmatrix} 1 & 1 \\ 1 & 0 \\ 0 & 0 \\ 1 & 1 \end{bmatrix}$$
$$[W] = \begin{bmatrix} 1 & 1 \\ 1 & 0 \\ 0 & 0 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 2 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 2 & 1 \end{bmatrix}$$

We have the weights calculated but still if the patterns are passed through the association matrix [W], it won't result into identical matrix. For the purpose, the values need to be passed through hard-limiter.

Let's consider one of the input [1101] is corrupted to [1111], still the correct result can be obtained as;

$$[1\ 1\ 1\ 1] \begin{bmatrix} 1 & 1 & 2 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 2 & 1 \end{bmatrix} = [2\ \ 3\ \ 5\ \ 3]$$

If 2 is the threshold value, the correct result is obtained: [0 1 1 1]. So, where does the Hopfield network come into all of this? The Hopfield network was based upon the learning matrix, with two changes - the Hopfield network uses bipolar values (-1, +1) instead of binary (0,1), and the feedback property of the network described earlier.

Now, back to the Network...
The response of a neuron is pretty much the same as in any network - if the net (weighted sum) is above a threshold, the neuron fires (in the Hopfield network, this is a +1), if it is below is outputs -1. Hopfield never stated what happens when net equals the threshold, so we assume that it stays the same.

Weights are calculated in much the same way as the learning matrix with a small difference. Since the outputs are the same as the inputs we don't need a [Y]. This is not all, though, since we do have to ensure that the output of a neuron cannot be associated with one of its inputs. This can be simply done by subtracting the unit matrix multiplied by the number of patterns (called [P]). Therefore, the final equation can be written:

$$[W] = [X]^t[X] - [P]$$

We also have to calculate thresholds. We just assume there is an additional neuron, which is used as an offset, and is permanently set a 1. We calculate them in the same way as weights, except since the output is always stuck at 1, the formula equates to:

$$[W_0] = [X]^t \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

Therefore, the thresholds in our example would be (remember, the binary values have now been converted into bipolar form):

$$[W_0] = \begin{bmatrix} 1 & 1 \\ 1 & -1 \\ -1 & -1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 2 \\ 0 \\ -2 \\ 2 \end{bmatrix}$$

Energy
The term *energy* when applied to neural networks really has nothing to do with energy is the physics sense. It is not quantified according to any known mean of energy as the real-world knows it. Energy in a neuron is a measure of how much stimulation is required to make the neuron fire. For example, if an input to a neuron has a huge weight (for example, 100) the neuron will most definitely fire, it does not take much energy at all to make this neuron fire. If the weight is a tiny number (for example, 0.0001) then it will take a lot of energy (or a lot of inputs) to make the neuron fire. Also, if a weight is negative, it acts as an inhibitor - thus, more energy is required. The energy on the entire network is just the sum of all the neuronal energies. Now, as in a lot of 'engineering' problems, we want to minimize the energy required to get a result. The exact same applies in this analogy.

Now, often in problems that neural networks are applied to there is a large problem space. Imagine that problem space as a series of hills and valleys, representing the energy of all solutions. Our goal is to get to the lowest point in the search space.

Now, to calculate the energy of a neuron, all other neurons have to stay constant (since their values are used). This is also true when updating the neuron values - therefore, it is important that the neurons are all updating **asynchronously** (not at the same time). In my

opinion, this slightly defeats the purpose of a neural network, since they're supposed to be parallel, nevertheless, the Hopfield network does its job well.

Therefore, most Hopfield network algorithms simply select a neuron at random and update it accordingly. After many iterations, the network should stabilize to one of the patterns it has learnt. It minimizes the energy and converges to a pattern its weights are set for. The network does have a drawback though. It will often converge to the local minimum, not the global minimum.

Imagine the series of hills and valleys described before. There are many hills and many valleys, each one of the valleys probably represents a 'decent' solution but the deepest valley is the best solution. The Hopfield network will often converge to its local minimum (the valley nearest to its starting point on a hill), not the best possible solution.

The energy, E, of the network;

$$E = -\frac{1}{2}\sum_{i,j} w_{ij} \; s_i \; s_j + \sum_i \theta_i \; s_i$$

The summary

- Generally considered to be fixed-weight models, they **don't learn**. However, one way to get the weights is through the supervised Hebbian outer-product summation. To be effective, the patterns should be reasonably **orthogonal**.
- N neurons, fully connected in a cyclic fashion:
    - Values are +1, -1.
    - Each neuron has a weighted input from all **other** neurons.
    - Weights are **symmetric**: $w_{ij} = w_{ji}$ and self-weights = $w_{ii} = 0$
    - Activation function on each neuron i is

      f(net) = sgn(net) = 1 if net > 0
      -1 if net < 0

      ($net_i = \Sigma w_{ij} x_j$)

        - If net = 0, then the output is the same as before, **by convention**.

- Thresholds/Biases
    - There are no separate thresholds or biases.
    - However, these could be represented by units that have all weights = 0 and thus never change their output.
- Activation function can be hard-limiter or continuous satlins (piecewise)

Detailed Discussion on Hopfield Network

03 – Hopfield.pdf (in images folder)

Brain-State-in-Box (BSB) Model

The model

The "brain-state-in-box" sounds like we have a brain in a box without body. But the model is defined as follows:

Let W be a symmetric weight matrix whose largest eigenvalues have positive real components. In addition, W is required to be positive semi-definite, i.e., $x^T W x >= 0$ for all x. Let x(0) denotes the initial state vector. The BSB algorithm is defined by the pair of equations:

$y(n) = x(n) + h\, Wx(n)$,
$x(n+1) = f(y(n))$.

Or more concisely, the updating rule of the "brain state" x (a vector) is

$x = f(x + h\, W\, x)$

where h is a small positive constant called the feedback factor. The f is a piecewise-linear function of the form

$f(x) = +1$ if $x > 1$;
$f(x) = x$ if $-1 < x < 1$;
$f(x) = -1$ if $x < -1$.

When the W is choosing with the required property (positivity of largest eigenvalues), the effect of the algorithm is to drive the system for components of x to binary values +1 or 1 for each of the neuron. We can view it as a mapping from continuous inputs x(0) to discrete binary outputs. The final states are of the form (-1,+1,-1,-1,+1,+1, ..., +1). This represents a corner of cube in an N-dimensional space of linear size 2, centered at origin. This is the box of the brain-state-in "a box". The dynamics is such that the state moves to the wall of the box and then drives to the corner of the box.

Energy function

There is a similar energy function for the BSB model given by

$E = -(h/2)\, S_{ij}\, w_{ij}\, x_i\, x_j$
$= -(h/2)\, x^T\, W\, x.$

It turns out much more general conditions exist to decide if an energy function exists. This is given in the form a Cohen-Grossberg theorem discussed in the textbook.

Application of BSB model

What is a good use of BSB model? A natural application for the BSB model is clustering. Such as the classification of radar signals from the source of emitters. The matrix W has to be (unsupervised) learned using some of the methods discussed in early chapters.

Boltzmann machines

Simulated Annealing

Annealing in metallurgy, a technique involving heating and controlled cooling of a material to increase the size of its crystals and reduce their defects.

SA algorithm replaces the current solution by a random "nearby" solution, chosen with a probability that depends on the difference between the corresponding function values and on a global parameter T (called the *temperature*), that is gradually decreased during the process. The dependency is such that the current solution changes almost randomly when T is large, but increasingly "downhill" as T goes to zero. The allowance for "uphill" moves saves the method from becoming stuck at local minima—which are the bane of greedier methods.

Steps:

1. Create initial solution
2. Assess solution
3. Randomly tweak solution (selecting neighbors)
4. Apply acceptance criteria
5. Reduce temperature
6. Repeat

Assessing solution – total energy = f(inputs). Example: N-Queens problem

Tweak solution – select nearest neighbors

Acceptance criteria – determined by probability of acceptance.

$P(\Delta E) = e^{-\frac{\Delta E}{T}}$

where, ΔE is the difference in energy before selecting the new solution and after selecting the new solution.

Reduce temperature - $T_{i+1} = \alpha \ T_i$

Boltzmann Machine

A Boltzmann machine is the name given to a type of stochastic recurrent neural network. Boltzmann machines can be seen as the stochastic, generative counterpart of Hopfield nets. They were one of the first examples of a neural network capable of learning internal representations, and are able to represent and (given sufficient time) solve difficult combinatoric problems. However, due to a number of issues discussed below, Boltzmann machines with unconstrained connectivity have not proven useful for practical problems in machine learning or inference. They are still theoretically intriguing, however, due to the locality and Hebbian nature of their training algorithm, as well as their parallelism and the resemblance of their dynamics to simple physical processes. If the connectivity is constrained, the learning can be made efficient enough to be useful for practical problems.

A Boltzmann machine, like a Hopfield network, is a network of units with an "energy" defined for the network. It also has binary units, but unlike Hopfield nets, Boltzmann machine units are stochastic. The global energy, *E*, in a Boltzmann machine is identical in form to that of a Hopfield network:

$$E = -\sum_{i<j} w_{ij} \ s_i \ s_j + \sum_i \theta_i \ s_i$$

Where:

- w_{ij} is the connection strength between unit *j* and unit *i*.
- s_i is the state, $s_i \in \{0,1\}$, of unit *i*.
- θ_i is the threshold of unit *i*.

The connections in a Boltzmann machine have two restrictions:

- No unit has a connection with itself.
- $w_{ij} = w_{ji}$ (All connections are symmetric.)

Boltzmann learning compares the input data distribution P with the output data distribution of the machine. The distance between these distributions is given by the Kullback-Leibler distance:

$$w_{ij}[n+1] = w_{ij}[n] - \frac{\partial G}{\partial w_{ij}}$$

where,

$$\frac{\partial G}{\partial w_{ij}} = -\frac{1}{T}[p_{ij} - q_{ij}]$$

Here, p_{ij} is the probability that elements i and j will both be on when the system is in its training phase (positive phase), and q_{ij} is the probability that both elements i and j will be on during the production phase (negative phase). The probability that element j will be on, p_i, is given by:

$$p_i = \frac{1}{1 + e^{\frac{-\Delta E_j}{T}}}$$

T is a scalar constant known as the temperature of the system.

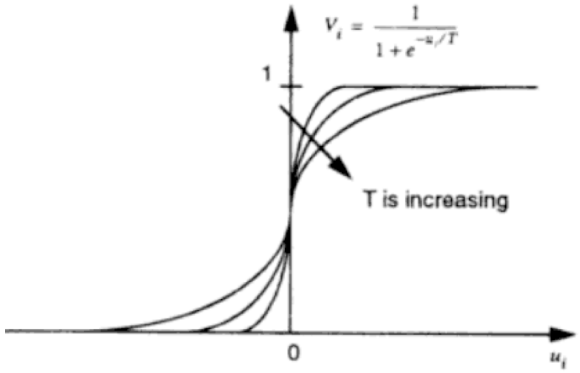

Fig. 4: Temperature of variations and effects in Boltzmann machine

Bi-Directional Associative Memory

Bidirectional Associative Memory, usually referred as **BAM,** is a type of recurrent neural network. There are two types of associative memory, auto-associative and hetero-associative. BAM is hetero-associative, meaning given a pattern it can return another pattern that is potentially of a different size. It is similar to the Hopfield network in that they are both forms of associative memory. However, *Hopfield nets return patterns of the same size*.

Topology

It contains two layers of neurons one we shall call X and Y. Layer X and Y are fully connected with each other. Once the weights have been established, input into layer X presents the pattern in layer Y, and vice versa.

Learning

Imagine we wish to store two associations, A1:B1 and A2:B2.

* A1 = (1, 0, 1, 0, 1, 0), B1 = (1, 1, 0, 0)
* A2 = (1, 1, 1, 0, 0, 0), B2 = (1, 0, 1, 0)

These are then transformed into the bipolar forms:

* X1 = (1, -1, 1, -1, 1, -1), Y1 = (1, 1, -1, -1)
* X2 = (1, 1, 1, -1, -1, -1), Y2 = (1, -1, 1, -1)

From there, we calculate M = Σ Xi'Yi.

Therefore, M =

2	0	0 -	2
0	-2	2	0
2	0	0	-2
-2	0	0	2
0	2	-2	0
-2	0	0	2

Recall

To retrieve the association A1, we multiply it by M to get (4, 2, -2, -4), which, when run through a threshold, yields (1, 1, 0, 0), which is B1. To find the reverse association, multiply this by the transpose of M.

References

Various sources.