

NEURAL COMPUTATION AND MEMORY

by

Yihe Lu

Dissertation for M.Sc. Mathematics

supervised by

Dr. Magnus RICHARDSON

August 28, 2014

Acknowledgements

I would like to express my deepest gratitude to my supervisor, Dr. Magnus Richardson. Long before the beginning of the dissertation when I first talked to him and showed my interest but innocence in neuroscience, he was very patient, introduced many interesting ideas, gave me a list of relevant modules as references, suggested me discussing with other experts in the area at Warwick and told me not to be anxious to decide a topic for dissertation. After taking his module on the introduction to theoretical neuroscience, I determined to carry out my dissertation under his supervision, even though I had just vague ideas about the particular area I wanted to study. Since starting the dissertation, I have received much more support from him. He recommended the very right book to start with, discussed with me the unexpected simulation results and even guided me in some details of academic writing. Most importantly, he has been trying to help me become an active researcher able to find answers and questions by myself.

I would also like to thank Dr. Hugo van den Berg, who brought me the initial interests in neuroscience when I was generally interested in mathematical biology and led me to Dr. Richardson, Dr. Yulia Timofeeva, Prof. Jianfeng Feng and Prof. Thomas Nichols, who gave me patient introductions on their various works in neuroscience in our early meetings, and my personal tutor, Prof. Roman Kotecky, who chatted with and encouraged me a lot on general academic topics and aims, in spite of his different expertise.

Finally I would like to dedicate my gratefulness to my families, my uncle and other relatives and my best friends. It has been their distant but warm encouragement that keeps me going on.

Abstract

This dissertation attempts to understand how a neural network could store and extract memories. Two typical artificial networks are to be mainly investigated. With an extremely simplified model of individual neurons, it is clear that memories are essentially stored in the synapses for both the networks. Hence, the most significant task is to built an appropriate connection matrix.

In the first one network, called the Hopfield model, the connection strengths are constructed according to the Hebb rule. Initial stability could be calculated by simple analysis but in order to study further the properties, the energy and the temperature are established, inspired by the networks similarity with the Ising model in statistical mechanics, and the memory capacity could thus be computed explicitly. However, the realistic application is not so satisfying as the theory suggest. It should be caused by the strong correlation in memories.

The second model is the layered feed-forward network, known as the perceptron. Due to its asymmetric connections, the techniques for the Hopfield model cannot be adapted. Nonetheless, it is possible to establish an error function as a global measure for the network to minimise while updating its connections. This procedure is performed by the back-propagation algorithm, which is widely used in supervised learning. Various alternatives for the algorithm are discussed and compared in the simulations. The results of the network learning are remarkable.

Contents

1	Introduction	5
1.1	Neuron as a Binary Threshold Unit	6
1.2	Network as a Uni-Directional Graph	7
1.3	Synapses as a Connection Strength Matrix	8
2	The Hopfield Model	9
2.1	The Basic Model	10
2.1.1	Connection Strength Construction	10
2.1.2	Crosstalk Term Analysis	12
2.1.3	Energy Function Approach	14
2.2	The Ising Model	15
2.2.1	Dynamics at Finite Temperature	16
2.2.2	Equilibria in Stochastic Networks	17
2.2.3	Capacity of Basic Hopfield	20
2.3	Simulation and Discussion	24
2.3.1	Algorithm Summary	24
2.3.2	Programming Details	24
2.3.3	Simulation Results	25
2.3.4	Crosstalk Revisit	29
2.4	Extension and Conclusion	30
3	Layered Feed-Forward Networks	31
3.1	One-layer Networks	33
3.1.1	Threshold Neurons	33
3.1.2	Linear Neurons	36
3.1.3	Nonlinear Neurons	38
3.1.4	Stochastic Neurons	39
3.2	Multi-Layer Networks	40
3.2.1	Scheme of Back-Propagation	41
3.2.2	Alternatives for Cost Function	44
3.2.3	Extensions of Learning Rate	45
3.3	Simulation and Discussion	47

3.3.1	Algorithm Review	47
3.3.2	Overfitting Problem	48
3.3.3	Toy Example	49
3.4	Conclusion and Application	54
4	Conclusion	55
	Appendices	61
A	Large Picture	61
B	MATLAB Code	62
B.1	The Hopfield Networks	62
B.2	The Perceptrons	64

1 Introduction

Memory is vital to intelligence, in either artificial or natural cases. Although a database is unable to draw conclusion by itself, a super computer able to defeat a human chess champion must be equipped with huge number of chess manuals. Given specific definitions or measures, one may argue that they are totally independent. Patients treated with bilateral hippocampal resection suffer grave loss of recent memory, but their intelligence is not affected, in the sense that they could pass some intelligence tests [1]. Nonetheless, it is an unfortunate fact that those patients are difficult to make real intellect contributions. The importance of memory is obvious. It is quite common that digital computers and human brains are considered as analogies to each other, but they do have various differences. Computers only outperform brains in tasks essentially based on simple arithmetic, while brains are flexible, able to tolerate errors and learn from failures, and very compact, able to compute in a highly parallel way but consume little energy [2]. However, it is still interesting to study artificial neural systems; at least it would be useful for computer science, and inspiration for neuroscience could be acquired through the artificial models.

Research on human brain offers different methods to classify and study memory. One may divide it into long-term and short-term, explicit and implicit [3], or retrospective and prospective [4]. These approaches are trying to understand memory in a descriptive way and thus mainly explain how the mechanism of memory works, but hardly what it essentially is. Therefore, the study of artificial memory systems is helpful, as it is constructed by hand and thus at least one of the variables in use must primarily be memory.

Two main similar features make an artificial model a reasonable analogy to the natural one: the model consists of units that are equipped with same computational function and the units have connections to others and in together form a network. It has actually become a convention in computer science to call the model a neural network as in neuroscience, and terms like neurons, synapses are directly adopted. This dissertation is devoted to artificial neural networks and uses such terms. Ideas directly linked to real neural networks will be pointed out in preventing any confusion.

1.1 Neuron as a Binary Threshold Unit

Real neurons are outstanding from other cells in many aspects. One of the most noticeable is their activity: they send out spikes (or action potentials), that is called firing. Spikes are typically non-linear phenomena as a neuron fires only if it receives stimulus large enough to make its membrane voltage go beyond its threshold. In 1943, inspired by this 'all-or-none' property, McCulloch and Pitts firstly applied propositional logic in the study of neural networks [5]. With little modification, the model can be summarised in one equation:

$$n_i(t+1) := \Theta \left(\sum_j w_{ij} n_j(t) - \mu_i \right) \quad (1)$$

See Figure 1 for the schematic of a neuron in the model. Here Θ is the Heaviside step function

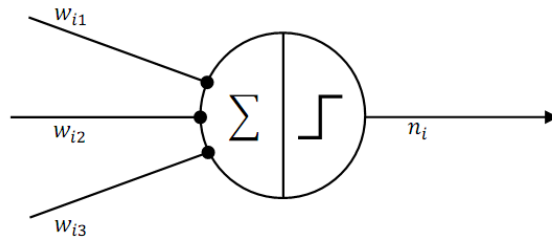


Figure 1: Schematic of a McCulloch-Pitts neuron: neuron i summarises information from weighted connections of other neurons and fires ($n_i = 1$) if the input goes beyond the threshold.

and $i, j = 1, 2, \dots, N$, assuming the network consists of N neurons. Here n_i is a binary variable of value either 1 or 0, w_{ij} is the connection strength onto neuron i from neuron j , and μ_i is the firing threshold of neuron i . Note the sign $:=$ is used to emphasise the updating process through time.

The binary values 1 and 0 of n_i represents the dual states of a neuron, firing and not firing, respectively. The connection strength w_{ij} could somehow be considered as the synaptic strength between neurons. The equation is actually an updating rule: neuron i summaries all information from neurons connected in the network and decides to fire or not according to its threshold.

Note all details of biophysics are extremely simplified into only two states and the remarkable morphology of neurons is represented by simple network topology. It could be a reasonable approach because neural networks usually have enormous sizes (the human brain has about 10^{11} neurons) and detailed, local and subcellular activities might not contribute too much to general, global and network functions. With these simplifications, a neuron becomes a binary threshold unit, which now makes the neural networks more comparable with computers.

A simple generalisation of Equation (1) is

$$n_i := g\left(\sum_j w_{ij}n_j - \mu_i\right) \quad (2)$$

where g , any sigmoid function varying between 0 and 1, replaces the Heaviside step function. n_i then could take any values in the interval if g is a continuous function. It is still possible in this case to consider a neuron as a binary unit but now n_i is the average activation. Certainly real neurons could also function continuously in some sense, but the core idea of the generalisation is to make it easier for analysis or computation; it might be dangerous to link the continuous property directly to real situations.

The generalised McCulloch-Pitts equation (2) is the basis of this entire dissertation. All the particular networks to be studied are made up of such binary threshold neurons.

In addition, for analytical and computational convenience, the binary values (or the lower and upper bounds of the interval in the continuous case) are taken -1 for not firing (very low activity) and 1 for firing (very high activity) in this dissertation, which is easily achieved by introducing a linear transformation; the forms of the above equations (and all those will follow) remain the same.

1.2 Network as a Uni-Directional Graph

Another unique characteristic of neural systems is its structure. A neuron has thousands of interconnections with other neurons, which allows signals to propagate through the network vastly. Within the brain cortex, there are three main classes of interconnections [6]:

1. feedforward connections: propagate input signals from one region to the given one, so they process information at an earlier stage;
2. recurrent synapses: interconnect neurons within a certain region, so along the processing pathway, the computational work is performed at this same stage;
3. top-down connections: send feedbacks from neurons at the later stage to those at the earlier one.

Although the McCulloch-Pitts model has ignored the spatial structure of the neurons, the connections are preserved as w_{ij} and thus the topology of the network is traceable. Note again w_{ij} means the connection from neuron j to neuron i . If it equals 0, it means neuron j cannot directly send information to neuron i . It is not necessary w_{ij} and w_{ji} are zero or non-zero at the same time, so the signal propagation is assumed uni-directional. The anatomy of real neurons

is similar to this model: dendrites receive information and axons send out signals. It is thus perfectly able to represent those three interconnection types. For a feed-forward network, if the indices of neurons runs against the flow of the information, that is, neurons process signals earlier are tagged with larger indices, w_{ij} shall be a lower triangle matrix; on the contrary, a top-down network shall have an upper triangle connection matrix. For a recurrent network, considering the large number of neurons each neuron contacts, the matrix shall be non-sparse (or dense).

1.3 Synapses as a Connection Strength Matrix

Synapses are the structures enabling neurons to pass on electronic or chemical signals to other cells; in the network of neurons only, they communicate with synapses. Several axon branches from one neuron may have contact with the same other one, a number of synapses may lie on the ends and each of them may behave differently in response to the same signal. Apparently, the model does not present all these biophysical details; instead, all synaptic contacts from one neuron to the other with their strengths and even temporal or spatial factors along the signal propagation are summarised all into w_{ij} . Hence, not only the network topology does w_{ij} as a matrix determines, but also the connection strengths between neurons.

In other words, the model is a weighted directional graph. If $w_{ij} > 0$, the firing of neuron j encourages that of neuron i , i.e. the effect is excitatory, similar for $w_{ij} < 0$ but with an inhibitory effect. The larger the absolute value w_{ij} is, the greater this effect is. Although it might be too rash to believe the properties of connection strengths are comparable with, or even derived from, those of synaptic strengths, the Hebb rule [7], which originated to explain adaptation of real neurons, is widely applied in artificial neural networks, often in a generalised version.

Adaptation occurs at all levels in the nervous system, from ion channels on synapses to individual neurons and entire structures of networks [8]. Among various forms it can take, activity-dependent synaptic plasticity is widely believed to be one of the most basic [6], which is well explained by the Hebb rule: if one neuron often contributes input to the firing of the other, then the synapses between them shall grow stronger. This would become a constructive prescription, or at least an inspiring suggestion, in deciding or designing the algorithm for the artificial networks to learn, i.e. to acquire, the appropriate values of w_{ij} . Actually, when the network is to be built able to learn from data by itself in Section 3, the only thing being renewed is the

connection strength matrix; the threshold terms can be absorbed, and there will be no attempt in any network to vary its size during the learning process. This hence implies the memory is essentially the values of w_{ij} , if the networks to be studied indeed appear to have memory.

As for learning, there are three types [9]: supervised learning, reinforcement learning and unsupervised learning. Supervised learning is training by a 'teacher'; a set of predetermined input-output pairs is imposed, the network adjusts its connection strengths until the desired computation emerges. Instead of a 'teacher' giving the correct relationships between inputs and outputs, reinforcement learning provides the network with feedbacks on its performance and it shall acquire the expected computation by optimising its utility or cost function, or something similar to evaluate the feedbacks. In unsupervised (or self-supervised) learning, there is no given right answers at all; the network is presented with data only and it shall learn solely by itself to function in a reasonable way, which is vague or too complex that the network designer is unable to define beforehand.

In summary, the general model to be studied here is an artificial network with neurons simplified as binary threshold units and all synaptic effects aggregated into a connection strength matrix. The size and the basic structure of the neural network, the activation function and the values taken by the neurons are determined before any analysis and computation. The only thing needs to be thoroughly studied is the connection strengths. Two specific networks of recurrent and feed-forward structures will be studied. The top-down network is merged into the feed-forward one as they share the symmetric structure. In aim of an appropriate connection strength matrix, different approaches are taken in the two networks. In the recurrent network, the matrix is constructed by hand, while the feed-forward network is designed able to acquire it by supervised learning. The general theories of these models are studied from Hertz et al. (1991) [2], and the genuine simulations and analysis performed follows mainly in the Simulation and Discussion sections for the two networks (see Section 2.3 and 3.3).

2 The Hopfield Model

Hopfield firstly introduced this model to neuroscience as an associative memory [10], which means the network is able to remember things in a fuzzy manner as a human brain does. To be specific, the network is able to respond to a presented pattern with the most similar memory

stored in it.

The model is a typical recurrent neural network with all N neurons connected to each other. As a complete graph, the connection strength w_{ij} should be almost non-zero. Any configuration ξ_i^μ is a possible pattern for the model to memorise, where $\mu = 1, \dots, p$.

2.1 The Basic Model

The neuron in this model takes binary values ± 1 , denoted by S_i instead. Hence, the Heaviside step function is replaced by the sign function $\text{sgn}(x)$, that is,

$$\text{sgn}(x) = \begin{cases} 1 & \text{if } x \geq 0; \\ -1 & \text{if } x < 0. \end{cases}$$

Now the McCulloch-Pitts Equation (1) becomes

$$S_i := \text{sgn}\left(\sum_j w_{ij} S_j - \theta_i\right)$$

and $\theta_i = 2\mu_i - \sum_j w_{ij}$ as $S_i = 2n_i - 1$. These threshold terms are all set as 0, because they are of no use with the random patterns to be considered. Thus the actual updating rule applied in this model is

$$S_i := \text{sgn}(h_i) \tag{3}$$

where $h_i = \sum_j w_{ij} S_j$ is the net input onto neuron i .

If the network functions as desired, reading an input pattern and adjusting states with this rule, the neural network shall finally present its memory that the most resembles the input. To achieve this aim, the memories themselves should firstly be stable and then if the network is also able to correct deviation from the memories, the model is then acceptable as a content-addressable memory system.

2.1.1 Connection Strength Construction

The connection strength matrix in this network is directly built using the condition of memory stability. For one memory ξ_i to be stable under the dynamic rule (3), the condition is, for all i ,

$$\text{sgn}\left(\sum_j w_{ij} \xi_j\right) = \xi_i \tag{4}$$

This is true if

$$w_{ij} \propto \xi_i \xi_j$$

as $\xi_j^2 = 1$. Apparently there can be other prescriptions, such as taking both ξ_i and ξ_j cubed, but the direct product of them is easier for analysis and computation. For later convenience, it is set that

$$w_{ij} = \frac{1}{N} \xi_i \xi_j \quad (5)$$

Note this directly makes the connection strength matrix symmetric.

So far, the connection strength is constructed in a straightforward manner. A satisfying consequence is that it naturally enables the network to self-correct errors. Given a different starting state S_i , The condition (4) holds as long as $S_i \neq \xi_i$ for some i fewer than half.

In other words, ξ_i is a natural attractor in the dynamics in the pattern space. States starting with more than half differences ends up in an other attractor at $-\xi_i$, which is called the reversed state. This phenomena occurs in all networks with binary nodes since the dual states are perfectly symmetric to each other. It shall not be problematic because one certain neuron could be chosen as indicator of the sign of all neurons; switching all neurons gives the expected result if the network does converge to the reversed state.

With the inspiration of storing one memory in the network, the simplest method to store multiple memories is just to make w_{ij} a linear combination of solutions (5) for every individual memorie μ , that is,

$$w_{ij} = \frac{1}{N} \sum_{\mu=1}^p \xi_i^\mu \xi_j^\mu \quad (6)$$

Note this is the Hebb rule but generalised. The original rule suggests the connection strength is changed in proportion to the correlation between the firing of the pre-synaptic neuron and that of the post-synaptic one, but Equation (6) goes further as the connection is also strengthened when neither of the neuron is firing.

Certainly there are models avoiding this generalisation [11], but it makes the model more complicated. In addition, a more general prescription [10] is

$$\Delta w_{ij} = f(V_i(t), V_j(t)) \quad (7)$$

which involves learning (Δw_{ij} is the change for every learning step) as w_{ij} is being renewed according to the temporal states of neurons i and j , in terms of V_i and V_j at time t , and the learning rule f is an arbitrary but appropriate calculation over the history of both neurons.

For instance, it is considered with the original Hopfield model that

$$\Delta w_{ij} = A \sum_s V_i^{s+1} V_j^s \quad (8)$$

where A is some constant and the superscripts denote time. Thus, this rule specifies the change of the connection strength between two neurons to be proportional to the superposition of products of their states with the post-synaptic one having a time delay, in order to reflect the causality along the firing sequence. This model makes w_{ij} nonsymmetric and may cause a memory to be only metastable in the sense that it will be replaced in time [10].

Various alternatives are possible and they could yield better results in solving particular problems, but they are beyond the scope of this dissertation.

2.1.2 Crosstalk Term Analysis

Back to the basic multiple memory prescription (6), exactly the same things are to be checked as for the one memory case. The stability condition now becomes, for all i ,

$$\text{sgn}(h_i^\nu) = \xi_i^\nu \quad (9)$$

Note here the subscripts again denote the pattern indices. The input is

$$h_i^\nu = \sum_j w_{ij} \xi_j^\nu = \frac{1}{N} \sum_j \sum_\mu \xi_i^\mu \xi_j^\mu \xi_j^\nu$$

which can be separated into the special term of $\mu = \nu$ and all the rest:

$$h_i^\nu = \xi_i^\nu + \frac{1}{N} \sum_j \sum_{\mu \neq \nu} \xi_i^\mu \xi_j^\mu \xi_j^\nu \quad (10)$$

The second term is called the crosstalk term. It is the interference from other memories with the memory ν and thus generates noise. If its magnitude is less than 1, the sign of h_i^ν is not changed and therefore the memories are stable. Let

$$C_i^\nu = -\xi_i^\nu \frac{1}{N} \sum_j \sum_{\mu \neq \nu} \xi_i^\mu \xi_j^\mu \xi_j^\nu \quad (11)$$

and then the net input and the multiple memory stability condition (9) can be written in a more compact form:

$$h_i^\nu = \xi_i^\nu (1 - C_i^\nu) \quad (12)$$

and if $C_i^\nu > 1$, the change of the sign of h_i^ν makes neuron i unstable in recalling memory ν .

In one memory case, there is no crosstalk term and thus the memory stored is always stable. Since

the pattern space has finite cardinality (2^N), the network can never store too many memories. Thus there must exist a maximum of stable memories, which apparently depends on ξ_j^μ . If random memories are assumed, to be specific, $\xi_j^\mu = \pm 1$ with either probability 1/2 and totally independence among all j and μ , then the probability $P_{\text{error}} = \Pr(C_i^\nu > 1)$ could be calculated. It is binomial distributed with mean 0 and variance $\sigma^2 = p/N$. Since Np is large, it is more convenient to approximate it with a Gaussian distribution of the same mean and variance:

$$\begin{aligned} P_{\text{error}} &= \frac{1}{\sqrt{2\pi}\sigma} \int_1^\infty e^{-x^2/2\sigma^2} dx \\ &= \frac{1}{2} [1 - \text{erf}(\sqrt{N/2p})] \end{aligned} \quad (13)$$

where the error function $\text{erf}(x)$ is defined by

$$\text{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x \exp(-u^2) du$$

It is now for the network designer to decide an acceptable performance, e.g., $P_{\text{error}} < 0.01$, and then the storage capacity p_{max} could be found accordingly.

However, there are three different explanations on being acceptable [2]:

1. $1 - P_{\text{error}} > 0.99$: requires individual neurons to recall almost correctly. Thus, $P_{\text{err}} < 0.01$, which gives $p_{\text{max}} = 0.15N$.
2. $(1 - P_{\text{error}})^N > 0.99$: demands each memory of a whole pattern to be recalled with few errors, thus the power of N . Since N/p is large, the asymptotic expansion of the error function, that is,

$$1 - \text{erf}(x) \rightarrow e^{-x^2}/\sqrt{\pi}x \quad \text{as } x \rightarrow \infty$$

could be used to approximate the probability, which gives

$$\begin{aligned} \log(P_{\text{error}}) &\approx -\log 2 - N/2p - \frac{1}{2} \log \pi - \frac{1}{2} \log(N/2p) \\ &< \log 0.01 - \log N = \log(0.01/N) \end{aligned}$$

Again since N is large, terms other than the leading ones could be omitted. The inequality thus becomes $N/2p > \log N$ which yields $p_{\text{max}} = N/2 \log N$.

3. $(1 - P_{\text{err}})^{pN} > 0.99$: insists all patterns to be memorised should be generally stable. Applying exactly the same trick in the case 2, the condition appears to be $N/2p > \log(Np)$, which gives $p_{\text{max}} = N/4 \log N$, because in the leading order $\log(Np) \sim \log N^2 = 2 \log N$.

Nonetheless, the capacity p_{max} only presents the stability condition at the first step of recalling. A small number of unstable initial neurons, which even is measured as acceptable by the above criteria, may lead more neurons to become unstable in consequence. The memory as a whole

might not be stable in such cases and therefore the current conditions obtained are necessary but insufficient; in other words, a smaller capacity should be expected. In aim of this, more powerful techniques will be applied.

2.1.3 Energy Function Approach

It is useful to define an Energy function (or a Hamilton) for the model [10]:

$$H = -\frac{1}{2} \sum_{ij} w_{ij} S_i S_j \quad (14)$$

The core property of the Hamilton is that it decreases monotonically as the dynamic evolves, that is, it reaches local minima of the energy surface when the network converges to some pattern.

To ensure the existence of the Hamilton, the connection strengths must be symmetric [2]. In this case, Equation (14) can be written as

$$H = H_0 - \sum_{i < j} w_{ij} S_i S_j$$

where $H_0 = \sum_i w_{ii}$ as $S_i^2 = 1$ for all i .

It is now straightforward to check the recalling algorithm (3) can only reduce the energy. Let S'_i be the updated value of S_i in accordance to the dynamics, that is

$$S'_i = \text{sgn} \left(\sum_j w_{ij} S_j \right) \quad (15)$$

Since the neuron takes only the binary values, either $S'_i = S_i$, then the energy remains unchanged; or $S'_i = -S_i$, in which case,

$$\begin{aligned} H' - H &= - \sum_{j > i} w_{ij} S'_i S_j + \sum_{j > i} w_{ij} S_i S_j \\ &= 2S_i \sum_{j > i} w_{ij} S_j \\ &= 2S_i \sum_j w_{ij} S_j - 2w_{ii} \end{aligned}$$

where the first terms is negative due to Equation (15) and $S'_i = -S_i$ and the second is also negative since $w_{ii} = p/N > 0$ given by the Hebb rule (6).

Actually it is set in the model that $w_{ii} = 0$. Though the value of these self-coupling terms does not affect on the stability as N is large, it is better to omit them because they do have impacts on the dynamics, producing additional stable spurious states [12].

It shall be constructed so that all memories locate exactly at the local minima. The starting point is again the one memory case, trying to build such a Hamilton that it is minimised when the input pattern $S_i = \xi_i$. A convenient choice with hindsight would be

$$H = -\frac{1}{2N} \left(\sum_i S_i \xi_i \right)^2$$

Similarly for many memories to store, let H be the composition over all patterns:

$$H = -\frac{1}{2N} \sum_{\mu=1}^p \left(\sum_i S_i \xi_i^\mu \right)^2 \quad (16)$$

which naturally makes all ξ^μ the local minima. Multiplying out the terms squared gives

$$H = -\frac{1}{2N} \sum_{\mu=1}^p \left(\sum_i S_i \xi_i^\mu \right) \left(\sum_j S_j \xi_j^\mu \right) = -\frac{1}{2} \sum_{ij} \left(\frac{1}{N} \sum_{\mu=1}^p \xi_i^\mu \xi_j^\mu \right) S_i S_j$$

which reproduces w_{ij} as directly constructed by the Hebb rule (6).

Now that it is confirmed that the energy function approach is essentially identical to the basic one, study of the Hamilton will directly offer understanding of the original dynamical system.

Before going further, it must be pointed out that ξ^μ are not the only attractors:

1. $-\xi^\mu$, the reversed states, are also local minima but not problematic as aforementioned;
2. ξ^{mix} , mixture states, are stable, different with any $\pm\xi^\mu$, but emerge as linear combinations of an odd number of memories [13];
3. for large p there are attractors that are not correlated with any stored pattern, called spin glass states sometimes [14].

In general, attractors of type 2 and 3 are the spurious minima. It would be ideal that they do not appear in the system, which is nearly impossible, but relatively small basins for these spurious minima are achievable.

2.2 The Ising Model

The construction of the energy function has implied that there is a noticeable analogy in physics for the Hopfield model. Actually the network is indeed isomorphic with an Ising model, which studies magnetic materials in statistical mechanics. The analogy becomes powerful when the neurons in the network is considered stochastic, which then brings in the idea of temperature that makes thorough analysis of the network dynamics more workable [2].

The simplest Ising model describes the magnetic material as a set of atomic magnets, called

spins, arranged on a lattice of crystal structure. Each spin is considered to point in two opposite directions, e.g. up or down, right or left, which are always denoted $S_i = 1$ or $S_i = -1$. Every spin i is affected by the magnetic field h_i at its own site,

$$h_i = \sum_j w_{ij} S_j + h^{\text{ext}}$$

which is made up of an external field h^{ext} and an internal field contributed by all the spins. Here w_{ij} is called exchange interaction strengths; they are symmetric in nature and the values depend on the particular magnet under study. The system also has an energy measure as

$$H = -\frac{1}{2} \sum_{ij} w_{ij} S_i S_j - h^{\text{ext}} \sum_i S_i \quad (17)$$

Now it is obvious that the Ising model is equivalent to the Hopfield model. The exchange interaction strengths of the magnet are the connection strengths in the network, the internal field over a spin is the net input onto a neuron and the external field works like the threshold.

2.2.1 Dynamics at Finite Temperature

At low temperature, a spin is mainly affected by the local magnetic field and points in its direction, that is $S_i = \text{sgn}(h_i)$, coinciding with the updating rule (3) in the neural network. However, when the temperature is not low, thermal fluctuations should be taken into account, which tend to flip the spin. Glauber dynamics [15] is commonly used to model this phenomenon. The deterministic updating algorithm is substituted by a stochastic rule:

$$S_i := \begin{cases} +1 & \text{with probability } g(h_i); \\ -1 & \text{with probability } 1 - g(h_i). \end{cases}$$

where $g(h)$ is a temperature-dependent function. The usual choice of the sigmoid function is

$$g(h) = f_\beta(h) = \frac{1}{1 + \exp(-2\beta h)} \quad (18)$$

where $\beta = \frac{1}{k_B T}$ and k_B is Boltzmann's constant. Note that

$$1 - f_\beta(h) = f_\beta(-h) \quad (19)$$

so the algorithm can be rewritten as

$$\Pr(S_i = \pm 1) = f_\beta(\pm h_i) = \frac{1}{1 + \exp(\mp 2\beta h_i)} \quad (20)$$

Note that for low temperature cases (in the large β limit), $f_\beta(h)$ reduces to a step function and thus the deterministic dynamics is recovered; in the opposite limit, the decision of S_i becomes purely random.

Now the idea of temperature is imposed onto the Hopfield model by applying the exactly model [16], applying Equation (18) or (20), but take $\beta = 1/T$, which is more convenient and has no effects because the temperature in the network is not a real physical measure.

2.2.2 Equilibria in Stochastic Networks

For a single spin to be in equilibrium, the average magnetisation of S , denoted by $\langle S \rangle$, can be calculated trivially:

$$\langle S \rangle = \Pr(+1) \cdot (+1) + \Pr(-1) \cdot (-1) = \tanh \beta h \quad (21)$$

where $\tanh(x)$ is the hyperbolic tangent function defined by

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

However, there is generally no analytic solution for many spin cases because the internal field is dependent on other random spins. A quite good approximation is available, known as mean field theory [2]:

$$\langle h_i \rangle = \sum_j w_{ij} \langle S_j \rangle + h^{\text{ext}}$$

which replaces the true values by their averages. Then calculation similar to the one spin case yields

$$\langle S_i \rangle = \tanh(\beta \langle h_i \rangle) = \tanh \left(\beta \sum_j w_{ij} \langle S_j \rangle + \beta h^{\text{ext}} \right)$$

The approximation becomes exact as the interaction range gets infinite [2].

For $p \ll N$, the neural network could directly adopt the equilibrium approximation as

$$\langle S_i \rangle = \tanh \left(\frac{\beta}{N} \sum_{j,\mu} \xi_i^\mu \xi_j^\mu \langle S_j \rangle \right) \quad (22)$$

whose solution is not obvious, but applying a trial solution that is proportional to one of the memories, the mean field equation becomes

$$\langle S_i \rangle = m \xi_i^\nu = \tanh \left(\frac{\beta}{N} \sum_{j,\mu} \xi_i^\mu \xi_j^\mu m \xi_j^\nu \right) \quad (23)$$

The terms in the summation could be separated into a term of ξ_i^ν and a crosstalk term as in Equation (10). Given the large N limit, the crosstalk term could be ignored, so the equation becomes

$$m\xi_i^\nu = \tanh(\beta m\xi_i^\nu)$$

Since $\tanh(-x) = -\tanh(x)$, it could be further simplified into

$$m = \tanh(\beta m) \tag{24}$$

which can be easily solved graphically (see Figure 2). Solutions are intersections between the

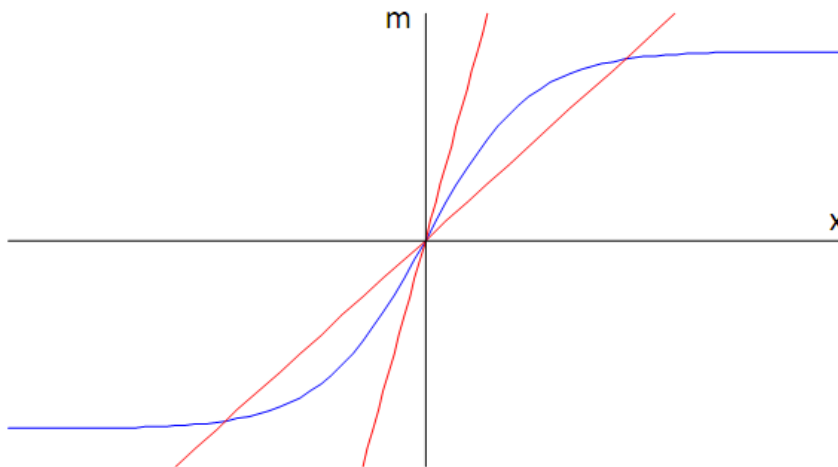


Figure 2: Solving Equation (24) by graph: the intersections of the blue curve $m = \tanh(x)$ and the red lines $m = x/\beta$ are the solutions.

sigmoid curve in blue $m = \tanh(x)$ and a family of red lines given by $m = x/\beta$. When $\beta \leq 1$, there is only one trivial solution at the origin, whereas $\beta > 1$ there are two more symmetric solutions.

The situation of only one trivial solution corresponds to the high temperature case, as β is small, which implies $\langle S_i \rangle = 0$ and thus neurons readjust their values in a completely random way. Note that the straight line is tangent to the sigmoid curve at the origin is a critical case, in which the temperature is called critical and denoted by T_c . In the stochastic network where $\beta = 1/T$, $T_c = 1$. In addition, T approaching 0 gives the large β limit, which makes the positive solution converges to 1 and the negative to -1 ; the deterministic model is recovered.

For the magnet analogy, it has Boltzmann's constant to scale the value, but this is of no more concern, because the Hopfield model has presented itself to be well-adapted with the stochastic model and the pseudo-temperature concept.

m is a local measure on a single neuron but a useful number to estimate the network behaviour by the relationship

$$m = \langle S_i \rangle / \xi_i^{\nu} = \Pr(i \text{ correct}) - \Pr(i \text{ incorrect})$$

and thus the average number of correct neurons in a pattern is calculable:

$$\langle N_{\text{correct}} \rangle = \frac{1}{2}N(1 + m) \quad (25)$$

Since the solution to Equation (24) is available, it is now straightforward to obtain the solution of $\langle N_{\text{correct}} \rangle$ (in accordance to the positive solution of m), which is presented in Figure 3. The

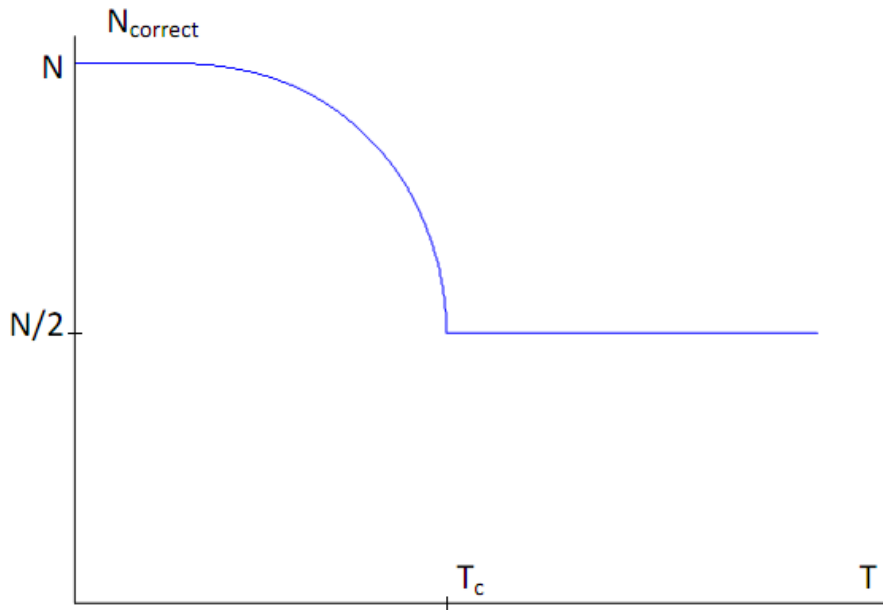


Figure 3: $\langle N_{\text{correct}} \rangle$ from Equation (25) as a function of the pseudo-temperature T

solution above T_c is $N/2$, which means the neurons are behaving in a purely stochastic way. Although the randomness increases dramatically when the temperature is increasing towards closely the critical value, the change is quite slow at low temperature.

At this range of relatively small T , the network behaviour is almost strictly correct as with the deterministic neurons. However, the stochastic dynamics is actually better, especially because the spurious minima are of high energy and thus network was stuck in such unexpected attractors is now easily kicked out by the randomness.

2.2.3 Capacity of Basic Hopfield

The above analysis has proved the stochastic model useful, but it is based on $p \ll N$ and thus of no help in finding the value of p_{\max} . However, it will be shown that the introduction of temperature is powerful and worth complicating the network with the randomness.

Now p has the same order as N , so the crosstalk term cannot be ignored as in Equation (23).

Define the crosstalk entry of pattern ν as

$$m_\nu = \frac{1}{N} \sum_i \xi_i^\nu S_i \quad (26)$$

which enables to rewrite the Hamilton (16) in a compact form as

$$H = -\frac{N}{2} \sum_{\mu=1}^p m_\mu^2 = -\frac{N}{2} \|m\|_2^2 \quad (27)$$

where $\|m\|_2$ is the l_2 -norm of the crosstalk vector $m = (m_\mu)_{\mu=1,\dots,p}$. Hence, the terminology is consistent with the crosstalk term defined in Equation (10).

Suppose the first pattern is under study, then m_1 is of order 1, while all other m_ν is small, of order $1/\sqrt{N}$, but the mean square overlap,

$$r = \frac{1}{\alpha} \sum_{\nu \neq 1} m_\nu^2 \quad (28)$$

where $\alpha = p/N$ is the load parameter, is of order 1, hence the crosstalk term is not negligible as expected.

Substitute $\langle S_i \rangle$ in Equation (22) into (26), with S_i replaced by $\langle S_i \rangle$, using the mean field theory again:

$$m_\nu = \frac{1}{N} \sum_i \xi_i^\nu \tanh \left(\beta \sum_\mu \xi_i^\mu m_\mu \right) \quad (29)$$

With this starting point, firstly study the case when $\nu \neq 1$. Separating the terms with $\mu = 1$ and $\mu = \nu$ gives

$$m_\nu = \frac{1}{N} \sum_i \xi_i^\nu \xi_i^1 \tanh \left[\beta \left(m_1 + \xi_i^\nu \xi_i^1 m_\nu + \sum_{\mu \neq 1, \nu} \xi_i^\mu \xi_i^1 m_\mu \right) \right]$$

In the argument of the hyperbolic tangent function, the first term is large by assumption, the third is large because there are about p terms, but the second is small of order $1/\sqrt{N}$, so it can be expanded using $\frac{d}{dx} \tanh(x) = 1 - \tanh^2(x)$ and explicitly it has become

$$\begin{aligned} m_\nu &= \frac{1}{N} \sum_i \xi_i^\nu \xi_i^1 \tanh \left[\beta \left(m_1 + \sum_{\mu \neq 1, \nu} \xi_i^\mu \xi_i^1 m_\mu \right) \right] \\ &+ \frac{\beta}{N} \sum_i \left\{ 1 - \tanh^2 \left[\beta \left(m_1 + \sum_{\mu \neq 1, \nu} \xi_i^\mu \xi_i^1 m_\mu \right) \right] \right\} m_\nu \end{aligned} \quad (30)$$

In order to continue the calculation, it has to be assumed now that the small overlaps m_μ for $\mu \neq 1$ are independent random variables; Equation (28) suggests their means to be 0 and variance $\alpha r/p$. At the same time, $\xi_i^\mu \xi_i^1$ in the second line is a random variable independent of m_μ , so by the central limit theorem the equation could be simplified as

$$\begin{aligned} m_\nu &= \frac{1}{N} \sum_i \xi_i^\nu \xi_i^1 \tanh \left[\beta \left(m_1 + \sum_{\mu \neq 1, \nu} \xi_i^\mu \xi_i^1 m_\mu \right) \right] + \beta m_\nu - \beta q m_\nu \\ &= \frac{N^{-1} \sum_i \xi_i^\nu \xi_i^1 \tanh [\beta (m_1 + \sum_{\mu \neq 1, \nu} \xi_i^\mu \xi_i^1 m_\mu)]}{1 - \beta(1 - q)} \end{aligned}$$

where

$$q = \int \frac{dz}{\sqrt{2\pi}} e^{-z^2/2} \tanh^2 [\beta(m_1 + \sqrt{\alpha r} z)] \quad (31)$$

To compute r , square m_ν . As only terms outside the tanh involves pattern ν , they could be averaged separately and only the self-coupling term survives, while the averaging over the tanh just yields a factor of q as in simplifying Equation (30). Comparing with Equation (28) gives

$$r = \frac{q}{[1 - \beta(1 - q)]^2} \quad (32)$$

Note the result is independent of ν .

Now use the exactly the same approach to study the case when $\nu = 1$, starting from Equation (29) again, an equation for m_1 is easily obtained:

$$m_1 = \int \frac{dz}{\sqrt{2\pi}} e^{-z^2/2} \tanh [\beta(m_1 + \sqrt{\alpha r} z)] \quad (33)$$

Now there are three equations (31), (32) and (33) in terms of m_1 , q and r , and they can be solved simultaneously.

Certainly the solutions could only be obtained numerically in general, but here it is possible to check the zero temperature limit (i.e. $\beta \rightarrow \infty$), which recovers the basic Hopfield model. It is clear that $q \rightarrow 1$ in this limit but $C \equiv \beta(1 - q)$ remains finite. The integrals in Equation (31) and (33) is evaluated by the following two identities:

$$\begin{aligned} & \int \frac{dz}{\sqrt{2\pi}} e^{-z^2/2} (1 - \tanh^2 \beta[az + b]) \\ & \simeq \frac{1}{\sqrt{2\pi}} e^{-z^2/2} \Big|_{\tanh^2 \beta[az+b]=0} \int dz (1 - \tanh^2 \beta[az + b]) \\ & = \frac{1}{\sqrt{2\pi}} e^{-b^2/2a^2} \frac{1}{a\beta} \int dz \frac{\partial}{\partial z} \tanh \beta[az + b] \\ & = \sqrt{\frac{2}{\pi}} \frac{1}{a\beta} e^{-b/2a^2} \end{aligned}$$

and

$$\begin{aligned}
& \int \frac{dz}{\sqrt{2\pi}} e^{-z^2/2} \tanh^2 \beta[az + b] \\
& \xrightarrow{T \rightarrow 0} \int \frac{dz}{2\pi} e^{-z^2/2} \operatorname{sgn}[az + b] \\
& = 2 \int_{-b/a}^{\infty} \frac{dz}{2\pi} e^{-z^2/2} - 1 \\
& = \operatorname{erf} \left(\frac{b}{\sqrt{2a}} \right)
\end{aligned}$$

The three equations can now be rewritten as

$$\begin{aligned}
C &\equiv \beta(1 - q) = \sqrt{\frac{2}{\pi\alpha r}} \exp \left(-\frac{m^2}{2\alpha r} \right) \\
r &= \frac{1}{(1 - C)^2} \\
m_1 &= \operatorname{erf} \left(\frac{m_1}{\sqrt{2\alpha r}} \right)
\end{aligned}$$

In aim of the capacity α only, a trick is to set $y = m_1/\sqrt{2\alpha r}$ and get

$$y(\sqrt{2\alpha} + \frac{2}{\sqrt{\pi}} e^{-y^2}) = \operatorname{erf}(y) \quad (34)$$

which could be solved graphically as shown in Figure 4. The red curve is the error function

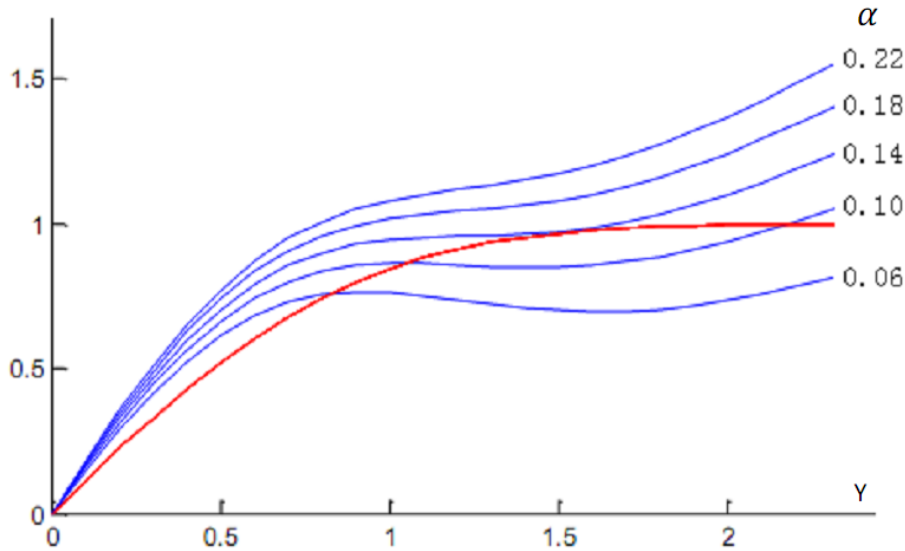


Figure 4: Solving Equation (34) by graph: the intersections of the red curve $z = \operatorname{erf}(y)$ and the blue ones $z = y(\sqrt{2\alpha} + \frac{2}{\sqrt{\pi}} e^{-y^2})$ are the solutions.

on the right hand side of the equation and the blues ones stands for the left hand side. Their intersections are the solutions and there is apparently a maximal value for α . This critical value for the existence of the solution can be computed numerically: $\alpha_c \approx 0.138$. The jump at α_c is

very dramatic, from $m_1 \approx 0.97$ to zero. From Equation (25), it is clear that the network could recall memories quite well for $\alpha < \alpha_c$ but becomes directly useless if α grows beyond the critical value.

The analysis of the deterministic dynamics only allows to conclude the initial capacity, but now with stochastic neurons, it becomes possible to study the property of the attractors directly, and the deterministic case is always easily recovered by taking zero temperature.

Amit et al. firstly calculated the complete solution and analysed the stable states further [14, 17]. The results can be presented in a phase diagram (see Figure 5). Networks in area A and B are

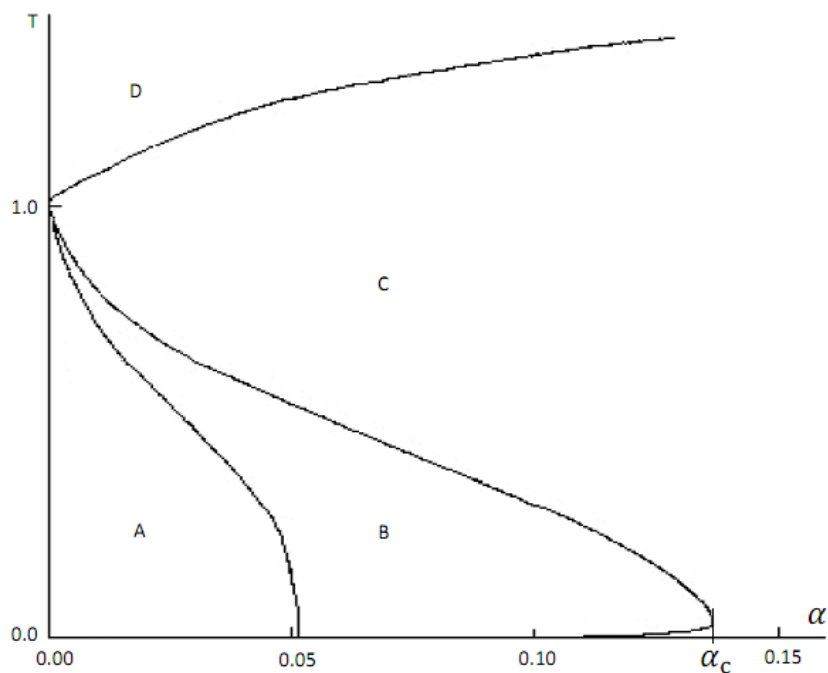


Figure 5: Phase diagram of the Hopfield model [14, 17]: networks of type A and B are useful memory systems, but spin glass states are more stable than stored memories in B; all attractors in C are spin glass states and only the trivial minimum zero occurs in D, and thus networks of type C and D are useless.

useful, because the stored patterns are indeed attractors, although in area B, spin glass states are more stable than memories. Systems of type B and C are of no use. All minima in C are spin glass states and zero is the only solution in D.

2.3 Simulation and Discussion

All the above theories are based on the assumption of random patterns, but realistic patterns cannot in general fulfil this condition. Simple but genuine simulations are performed to test whether or not the neural network yields an acceptable results as expected.

2.3.1 Algorithm Summary

The simulation applies the basic Hopfield model. The neural network is firstly presented with patterns to be memorised. The connection strength matrix is computed accordingly and then the network is ready.

A test pattern is now presented to the prepared network. It begins to recall the memory most similar to the test pattern by the updating rule (3). This process could be performed synchronously, but a central clock is then needed, and thus it becomes sensitive to time errors [2]. Therefore, an asynchronous approach is applied more commonly, which is more natural, accurate and convenient to use in practice.

There are two possible proceeding approaches to realise this recalling process. Each neuron could readjust its state at a random time but with a mean attempt rate, or a randomly chosen neuron updates its state at each time step. They are apparently equivalent to each other (with many neurons and long time, which is easily fulfilled in neural networks) because the probabilities of a neuron not readjusting in two cases are geometric and exponential decays respectively; the two distributions converge to each other. It is advantageous for hardware design to apply the first strategy, performing parallel computation as real nervous systems, but for simulations, the second idea is more practical.

The network is to stop after predetermined time epochs, long enough for the system to converge into some attractor, not necessary the desired memory, though.

2.3.2 Programming Details

The programming is done by MATLAB R2011b without any additional toolbox on a win32 system. Due to the memory limit, a picture with 100×100 is already out of the calculation capacity.

Patterns used here are 2-D pictures with pixels blacks and white only. The file format is chosen

or is standardised to be BMP. Thereby, a bit, or a pixel, is treated as a neuron in the Hopfield model. When a neuron is firing, the pixel is black with value 255; when a neuron is not firing, the pixel is white with value 0.

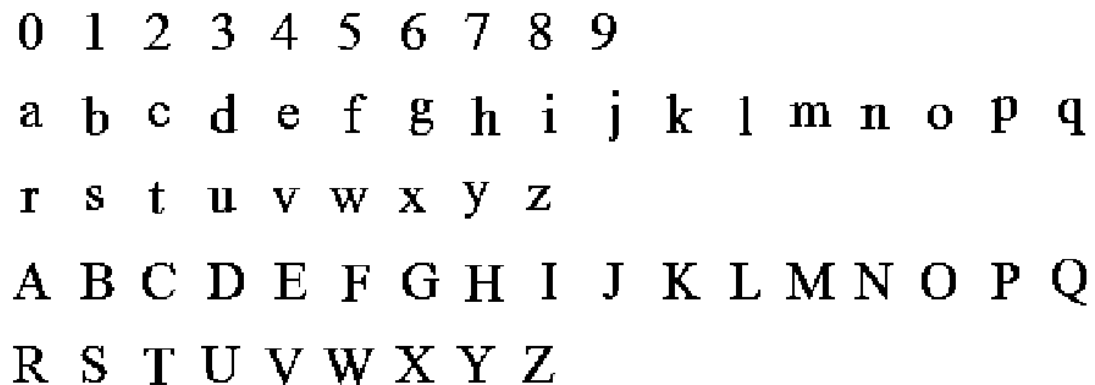
Since the Hopfield model has no growth or deterioration of neurons in the network, the size of pictures should be consistent for the same learning and recalling process. Pictures with 30×30 pixels and 80×80 are used.

The patterns chosen are digital numbers, English letters and Chinese characters.

2.3.3 Simulation Results

Simulation 1: Digits and Letters

10 digits and 26 pairs of English letters in their upper and lower cases are presented in the font of Times New Roman and their sizes are 20 (see Figure 6). Then the 62 pictures in total are



0 1 2 3 4 5 6 7 8 9
a b c d e f g h i j k l m n o p q
r s t u v w x y z
A B C D E F G H I J K L M N O P Q
R S T U V W X Y Z

Figure 6: Images of the digits and letters used in the simulations

standardised to BMP's with 30×30 pixels, black and white only.

These images are then read and stored in a 3-D matrix by Code H1, the connection strengths are calculated by Code H2 and the recalling is performed by Code H3 (see Appendix B.1 for details).

The results is disappointing. Whatever the input is, the pattern fades away; the network always converges to the trivial attractor where all neurons share the same state.

Simulation 2: Digits only

In order not to enlarge the network size but to have a relatively clearer memory, trying to store less patterns is a computationally economic way. With easy modifications on Code H1, the same

procedure as in Simulation 1 can be operated.

Some results are shown in Figure 7. The first column corresponds to the original test input and

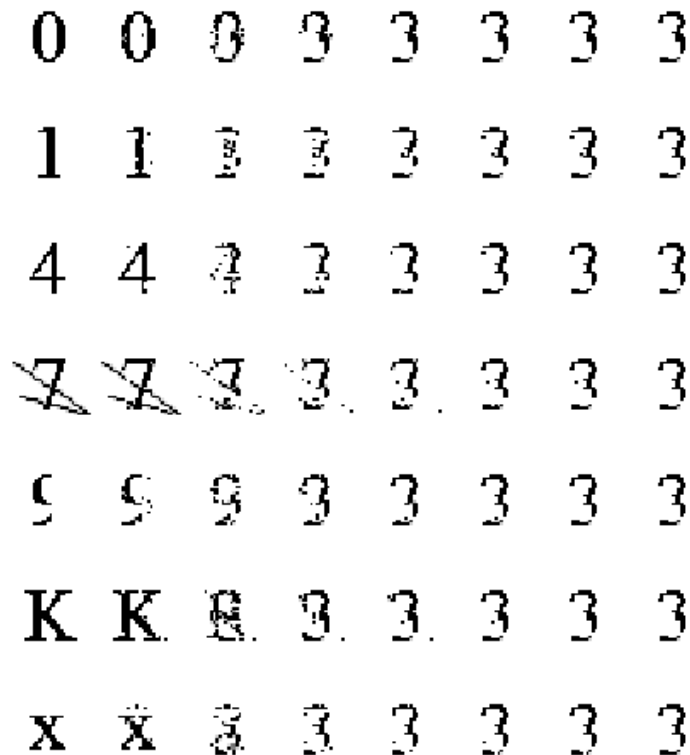


Figure 7: Results of the simulations with desired memory of digits only: the first column presents the test inputs and from the second to the last are readjustments after 100, 1000, 2000, 3000, 4000, 5000 and 10000 times; all test inputs evolves into the same attractor.

then the columns from the second to the last are outcomes after updates after 100, 1000, 2000, 3000, 4000, 5000 and 10000 epochs.

It seems whatever the input is, the network ends up at the same attractor eventually. Note the first three rows are the cases in which the original inputs are exactly the same as the stored memory. Therefore the memory is not stable at all, that is, the network is not a good memory system.

One may argue the attractor looks like the digit 3 and it might have some special contribution to the network. However, the simulation without '3' as the given memory yields the similar results (see Figure 8). The 3-like pattern is more likely to be the combination of all the digits, that is, the crosstalk term is too large.

Hence, the network as a memory system is useless, although it is improved a little better with less memories to store.

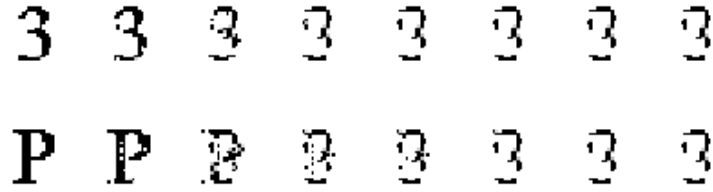


Figure 8: Results of the simulations with desired memory of digits without 3: the attractor is similar to that in Figure 7.

According to the theory of the Hopfield model, the network (with 900 neurons) should be at most able to store about $135(= 900 \times 0.15)$ patterns. The realistic network cannot even store 10 memories. Two possible factors are considered to cause this failure. First, the random memory assumption is not justified, which makes the crosstalk term much larger than expected. Second, too much margin of the pictures allows too few neurons to store the effective information, which reduces the network capacity.

Simulation 3: Chinese Characters

It might be helpful to use Chinese characters as the memories. As a type of logograms, there are a consider number of symbols and only ten common characters are taken (see Figure 9), which

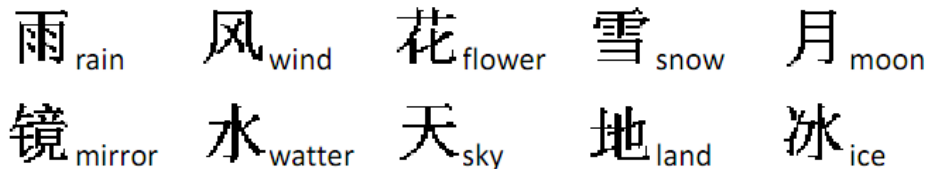


Figure 9: Images of the Chinese characters used in the simulations

should reduce the correlation among patterns; note no characters share the similar symbolic part except the ones of water and ice. Additionally, as symbols most fit in a square, they span more pixels inside an image, which seems to be an advantage comparing to the digits. At last, the patterns are still realistic and meaningful to people.

All the patterns are still 30×30 . The font applied is SimSun and the size is 20. Code H1, H2 and H3 can be easily adapted.

However, the network behaves unacceptably in a similar way as Simulation 2 (see Figure 10): the memories themselves are unstable and all test patterns are captured by the same attractor.



Figure 10: Results of the simulation with the Chinese characters: the first column presents the test inputs and from the second to the last are readjustments after 100, 1000, 2000, 3000, 4000, 5000 and 10000 times; all test inputs evolves into the same attractor.

Simulation 4: Large Chinese Charaters

It seems the random memory assumption is hard to justify. Although using Chinese characters may have reduced correlation between patterns, all pictures are clear to have locally correlated features. A larger network might improve the performance.

Same characters are chosen but now the size is enlarged (of size 72) and the font applied is STXinwei as shown in Figure 11. Similarly, Code H1, H2 and H3 could be easily adapted.

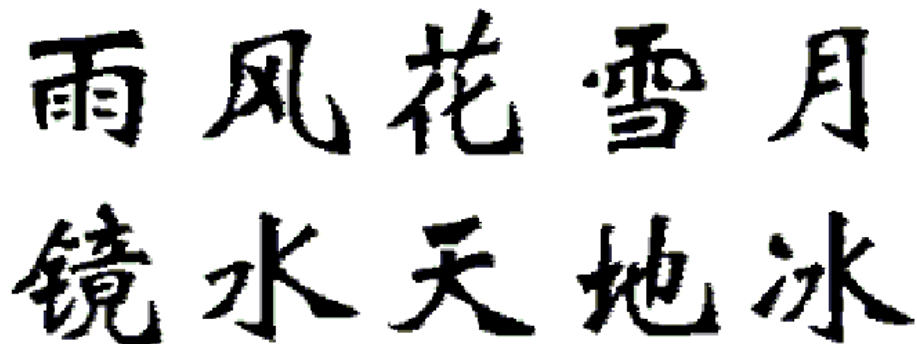


Figure 11: Images of the large Chinese characters used in the simulations

Unfortunately, the network is still of no use (see results in Figure 21 in Appendix A) even with a memory load of $10/6400 (= 0.0015625)$. It is now hard to argue that there are too less effective

neurons so that the large N assumption is not justified. The failure of the random memory assumption must take the main responsibility of the disappointing performance.

2.3.4 Crosstalk Revisit

To understand the unexpected results from the simulations, the crosstalk term has to be studied in more details. The compact form of Hamilton (27) is easier to work with.

Without loss of generality, consider how to ensure ξ^1 being a local minimum. For any S with $S_i = \xi_i^1$ for $i \neq j$ and $S_i = -\xi_i^1$ for $i = j$, $H(\xi^1)$ is a local minimum if $H(\xi^1) \leq H(S)$ for all $j = 1, \dots, N$.

$$\begin{aligned} H(S) - H(\xi^1) &= -\frac{N}{2} \left(\sum_{\mu=1}^p (m_\mu(S))^2 - \sum_{\mu=1}^p (m_\mu(\xi^1))^2 \right) \\ &= -\frac{N}{2} \sum_{\mu=1}^p \left(m_\mu(S) + m_\mu(\xi^1) \right) \left(m_\mu(S) - m_\mu(\xi^1) \right) \end{aligned}$$

Applying $m_\mu(S) - m_\mu(\xi^1) = -\frac{2}{N} \xi_j^1 \xi_j^\mu$ and $m_\mu(S) + m_\mu(\xi^1) = 2(m_\mu(\xi^1) - \frac{1}{N} \xi_j^1 \xi_j^\mu)$, to simplify the equation gives

$$H(S) - H(\xi^1) = \frac{2}{N} \sum_{\mu=1}^p \left(N \xi_j^1 \xi_j^\mu m_\mu(\xi^1) - 1 \right)$$

Hence, to make ξ^1 stable, the condition is

$$N \sum_{\mu=1}^p \xi_j^1 \xi_j^\mu m_\mu(\xi^1) \geq p \tag{35}$$

When $p = 1$, the inequality holds trivially, since $\xi_j^1 \xi_j^1 = m_1(\xi^1) = 1$.

When $p = 2$, the inequality becomes

$$N \geq 2 - N \xi_j^1 \xi_j^2 m_2(\xi^1)$$

It would be invalid, only if $\xi_j^1 \xi_j^2 m_2(\xi^1) < 0$ and $|N m_2(\xi^1)| = |\sum_i \xi_i^1 \xi_i^2| > N - 2$. It is nearly impossible to violate the second condition because it means the two patterns only have one unit different (or only one unit the same).

When $p \geq 3$, the inequality (35) become

$$N \geq p - N \xi_j^1 \xi_j^2 m_2(\xi^1) - N \xi_j^1 \xi_j^3 m_3(\xi^1) - \dots - N \xi_j^1 \xi_j^p m_p(\xi^1)$$

In the cases of only one or two memories, the simulations indeed yield perfect performance. However, for three memories, the network immediately becomes useless. In this case, the condition

is

$$N \geq 3 - N\xi_j^1\xi_j^2m_2(\xi^1) - N\xi_j^1\xi_j^3m_3(\xi^1)$$

while the three patterns of digits '1', '2' and '3' share 747 exactly the same pixels. The right hand side could be greater than N if $\xi_j^1\xi_j^2 = \xi_j^1\xi_j^3 = -1$, whose probability is not negligible (1/4 if assume independence between patterns). Apparently this probability would grow with p increasing.

2.4 Extension and Conclusion

As aforementioned, there are many alternative constructions and extension for the details in the Hopfield model, mostly varying w_{ij} in correspondence to ξ_i^μ of certain informative structure.

Two studies might be particularly of interest concerning the simulations with digits and characters have been done. The first one is of sparse patterns. The minor fraction of black pixels on the white screen is a typical example of this case. The connection strengths shall be

$$w_{ij} = \frac{1}{N} \sum_{\mu} (\xi_i^{\mu} - a)(\xi_j^{\mu} - a)$$

where $a \ll 1$ is the proportion of firing neurons in the memory and an optimal threshold (same for all neurons) should be chosen [18]. Due to the sparsity, the network can store a relatively larger number of memories with

$$\alpha_c \approx \frac{1}{2a|\log a|}$$

Note ξ_i^μ is still assumed independent for all i and μ but the probability for firing or not is biased. Nonetheless, simulation results are not improved by this technique, which re-emphasises the significance of the random memory assumption.

The second extension is the analysis on correlated patterns. Lowe [19] considered two types of correlation, semantic and spatial. The semantic correlation means $(\xi_i^\mu)_{\mu=1,\dots,p}$ is a homogeneous Markov chain but independence is still assumed within one pattern, while the spatial correlation represents $(\xi_i^\mu)_{i=1,\dots,N}$ being so but patterns are uncorrelated. It is proved that, the memories, in both cases, are almost surely stable.

Although this result cannot help enhance the capacity of the neural network, it implies that the patterns presented to the network have a more complicated structure in the crosstalk term, which neither the assumption of complete randomness nor the construction of Markov chain could explain. The crosstalk term is thus of more interest and importance in the study of the

Hopfield model, but it should be more concerned with the particular problem one wants to solve, which will go beyond the scope of this dissertation.

To conclude, the Hopfield model is a recurrent neural network, which works well if the crosstalk is under control. However, the simulations have shown that realistic patterns, at least images, frequently have crosstalk structures out of a general control. It seems not a good idea to directly apply the model in pattern recognition; informative insights and other techniques should be combined. Since recurrent synapses in real nervous systems generally locate at later stages along a information processing pathway, patterns such networks deal with may contain less correlation as in a picture; the Hopfield model might thus be an appropriate generalisation as the theories suggests.

3 Layered Feed-Forward Networks

Layered feed-forward networks, also known as perceptrons, is initially invented by Rosenblatt in 1957 [20], which produced one of the very first artificial neural networks in the world. The model is designed to be highly structured, consisting of several hierarchical layers, and information can only flow from one layer to the next one (an example is shown in Figure 12). Note the

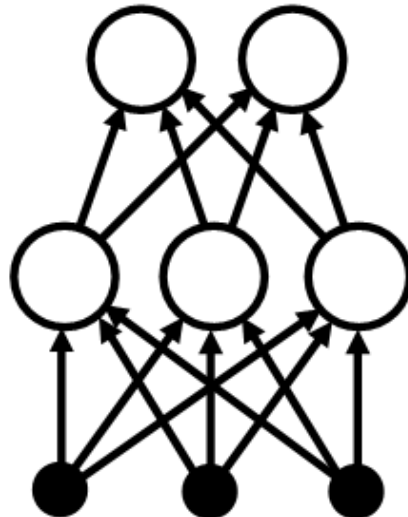


Figure 12: An example of a two-layer perceptron: the solid circles represents the input terminals, which are not counted as a layer; the circle at the top layer represents the output neurons and those in the middle form the hidden layer; the arrows show the information follow of the connections.

architecture of real neural networks is generally layered as well and one of the most common is the feed-forward network [2]. However, it is not likely to be strictly layered as the artificial networks, but may include loops, which makes the network partially recurrent.

Perceptrons are designed to consist of neurons that could be grouped into subsets. The first one directly receives signals from input patterns and send them to the rest part of the network; these neurons are thus called input terminals (solid circles in Figure 12). The second part is made up of several hidden (or intermediate) layers. The last layer is called the output layer as its neurons shows the final result of the network computation. Only the hidden and output layers are counted as the layers of perceptrons, as the input terminals are not equipped with the same computational ability as the other neurons; there is the other convention which does count it, though. Note there could be no hidden layers; in such simplest cases, there is only one layer and these networks are known as simple perceptrons.

The desired computation is similar as for the recurrent networks. Given an input patterns ξ , the network is expected to yield the correct match ζ . Hence, the task is also to establish an appropriate a connection matrix w_{ij} . However, due to the hierarchical structure and the unidirectional signal propagation, feed-forward networks have asymmetric w_{ij} with all entries above the diagonal being zero, which makes it unable to apply the energy function and all other methods and analysis based on it.

To find the appropriate connection strengths, the technique of supervised learning is applied in this content. Pairs of (ξ^μ, ζ^μ) are presented to the network. After training, it is expected to recall the memory ζ^μ given the input ξ^μ ; furthermore, for patterns different to ξ^μ but similar, it would be satisfying if the network could still present ζ^μ . By achieving this aim, the feed-forward network, though artificial, mimics the learning behaviour of the natural neural networks.

Note w_{ij} will be re-structured into non-square matrix in general from now on, in order to save notational and computational cost. w_{ij} still denotes the connection strength from neuron j to i , but since the network is strictly unidirectional, neuron i always locates on the next layer of neuron j ; i and j will then never represent the same neuron even if $i = j$, as they run through different indices depending on the size of the layers.

3.1 One-layer Networks

By definition, a simple perceptron has only one layer, the output layer, and no hidden ones. Since the sizes of input and output patterns are given, the structure of the network is determined accordingly. The computation is again based on the generalised McCulloch-Pitts equation (2):

$$O_i = g(h_i) = g\left(\sum_{k=1}^N w_{ij}\xi_k - \theta_i\right) \quad (36)$$

where the input terminals directly adopt the notation for patterns as ξ_k and O_i denotes the output. The threshold θ_i could be absorbed by setting $\xi_0 = -1$ and $w_{i0} = \theta_i$, which gives

$$O_i = g(h_i) = g\left(\sum_{k=0}^N w_{ik}\xi_k\right) \quad (37)$$

Similarly, it is the connection strength matrix essentially defines the network computation, so the aim is the same: to obtain w_{ij} that enables the network to match the actual output pattern with the desired, that is,

$$O_i^\mu = \zeta_i^\mu \quad (38)$$

for each i and μ .

3.1.1 Threshold Neurons

The simplest case would be a perceptron with deterministic binary threshold neurons. To be specific, $O_i^\mu, \xi_i^\mu, \zeta_i^\mu \in \{1, -1\}$ and $g(h) = \text{sgn}(h)$.

In order to fulfil the condition (38), the net input h_i^μ should have the same sign as that of ζ_i^μ for each i and μ , that is, for every μ

$$\text{sgn}(\mathbf{w} \cdot \boldsymbol{\xi}^\mu) = \zeta^\mu \quad (39)$$

Note the subscript i is dropped because it is convenient to consider only one neuron at a time, and that the connection strength and the input pattern are in the vector form. This means that the plane $\mathbf{w} \cdot \boldsymbol{\xi} = 0$ divides the space of the pattern vectors into two types of positive and negative outputs.

By defining $\mathbf{x}^\mu = \zeta^\mu \boldsymbol{\xi}^\mu$, Equation (39) becomes

$$\mathbf{w} \cdot \mathbf{x}^\mu > 0 \quad (40)$$

This means that the vectors \mathbf{x} must all lie on the same side of the plane $\mathbf{w} \cdot \mathbf{x}^\mu = 0$.

This property is called the linear separability. However, this plane does not always exist. If the

given patterns do not have the property, the network can never learn to perform the desired computation.

For example, the Boolean AND function is linearly separable. There are generally two types of failure (see Figure 13). The first type is represented by the Boolean XOR function. The second

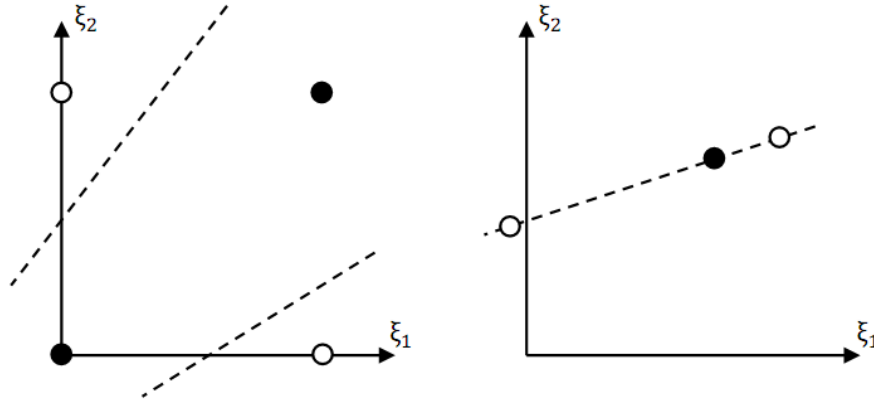


Figure 13: Two cases without the linear separability [2]: the XOR problem (left) and the linear dependence case (right).

type is caused by linear dependence of patterns.

Assume linear separability, there is a simple learning algorithm, which can be realised with different detailed models:

$$\Delta w_{ik} = \begin{cases} 2\eta \zeta_i^\mu \xi_k^\mu & \text{if } \zeta_i^\mu \neq O_i^\mu \\ 0 & \text{otherwise} \end{cases} \quad (41)$$

or

$$\Delta w_{ik} = \eta(1 - \zeta_i^\mu O_i^\mu) \zeta_i^\mu \xi_k^\mu \quad (42)$$

or

$$\Delta w_{ik} = \eta(\zeta_i^\mu - O_i^\mu) \xi_k^\mu \quad (43)$$

where η is called the learning rate.

In spite of different forms, the three updating rules share the same core idea. At a certain step of learning, if the output coincides with the input, $\zeta_i^\mu = O_i^\mu$ for all i , $\Delta w_{ik} = 0$ since it is not necessary to readjust the connection strength; otherwise, the generalised Hebb rule is applied, that is, the connection is strengthened in proportion to the product of pre- and post-synaptic neuron states. In the case of Equation (43), one may consider the network is trying to forget the incorrect matches (to learn the opposite) while learning the desired.

Note this product directly interacts with the learning rate, which decides the 'speed' of learning. Large η assigns more weight on the patterns newly learnt; w_{ij} might oscillate. On the other hand, small η readjusts w_{ij} in a humble manner. Both situations may lead to slow convergence to effective connection strengths. An appropriate choice of the learning rate is thus important. Sometimes, it is a better idea to require the value of the output larger than some given number

$$\mathbf{w} \cdot \mathbf{x}^\mu > N\kappa \quad (44)$$

where κ is called the margin size. As the sum on the left-hand side scales with N , placing N on the right-hand side could keep κ fixed.

Applying this new rule to equation (42) gives the perceptron learning rule [21]:

$$\Delta w_{ik} = \eta \Theta(N\kappa - \zeta_i^\mu O_i^\mu) \zeta_i^\mu \xi_k^\mu \quad (45)$$

or, in the vector form,

$$\Delta \mathbf{w} = \eta \Theta(N\kappa - \mathbf{w} \cdot \mathbf{x}^\mu) \mathbf{x}^\mu \quad (46)$$

Note Equation (42) is the special case with $\kappa = 0$ and the scaling in η of factor 2.

Equation (46) gives a geometrical description of the updating rule. If the projection of the connection strength matrix onto \mathbf{x}^μ , i.e. $\mathbf{w} \cdot \mathbf{x}^\mu$, is less than $N\kappa/|\mathbf{w}|$, \mathbf{w} is then changed in the direction of \mathbf{x}^μ . This procedure iterates until all projections are large enough. It is proved that this algorithm ensures w_{ij} to converge to the optimal solution in finite steps [21].

A measure to evaluate how easy or hard the problem arises according to the margin:

$$D_{\max} = \max_{\mathbf{w}} D(\mathbf{w}) \quad (47)$$

where

$$D(\mathbf{w}) = \frac{1}{|\mathbf{w}|} \min_{\mu} \mathbf{w} \cdot \mathbf{x}^\mu \quad (48)$$

is the smallest margin for all the given patterns, that is, the distance between the plane perpendicular to \mathbf{w} and \mathbf{x}^μ that is the nearest to it. The value is only determined by the direction of \mathbf{w} due to the factor of $1/|\mathbf{w}|$. Apparently, if the nearest pattern vector could have a positive margin, then the linear separability holds.

Hence, if this worst-case margin is maximised as in Equation (47), the solution, called optimal perceptron, naturally represents the solvability of the problem. If $D_{\max} < 0$, there is no solution. If $D_{\max} \geq 0$, there is a solution and the larger the value is, the easier the problem. For instance, for the Boolean AND problem, $D_{\max} = 1/\sqrt{17}$, while, for XOR, $D_{\max} = -1/\sqrt{2}$.

3.1.2 Linear Neurons

From now on the activation function $g(h)$ is generalised to be continuous and differentiable. This technique is useful because it allows the system to be equipped with a cost (or error) function $E(\mathbf{w})$, which directly measures the network performance. Then through the minimisation of $E(\mathbf{w})$, the appropriate \mathbf{w} is obtained.

Here assume the neuron is linear, that is, $g(h) = h$, which makes it easier to analyse, though less useful in practice comparing to non-linear neurons. Remark that image compression is an important application of linear networks, where it is proved theoretically [22] and indeed found by the simulations [23] that nonlinearity in this content is of no advantage.

Given input ξ_k^μ , a linear simple perceptron is to present the output as

$$O_i^\mu = \sum_k w_{ik} \xi_k^\mu \quad (49)$$

With the usual association condition (38), w_{ik} could be explicitly solved by the pseudo-inverse method [12]:

$$w_{ik} = \frac{1}{N} \sum_{\mu\nu} \zeta_i^\mu (\mathbf{Q}^{-1})_{\mu\nu} \xi_k^\nu \quad (50)$$

where

$$\mathbf{Q}_{\mu\nu} = \frac{1}{N} \sum_k \xi_k^\mu \xi_k^\nu \quad (51)$$

or, recalling Equation (26),

$$\mathbf{Q}_{\mu\nu} = m_\nu(\xi_\mu) \quad (52)$$

which is just the crosstalk entry between input patterns μ and ν .

Note O_i^μ is now continuous-valued, but it is still possible to restrict the target ζ_i^μ to ± 1 . The condition for a solution to be available is the existence condition of \mathbf{Q}^{-1} , which requires the inputs are linear independent.

It should be pointed out that linear independence implies the linear separability but the reverse is invalid; thus, it is a sufficient but not necessary condition for the existence of a solution [2].

Although an explicit solution could be calculated, it is more interesting to find a learning rule as for the threshold perceptrons. This effort will become valuable when non-linear neurons are under study.

As promised, a cost function is defined at the first place:

$$E(\mathbf{w}) = \frac{1}{2} \sum_{i\mu} (\zeta_i^\mu - O_i^\mu)^2 = \frac{1}{2} \sum_{i\mu} \left(\zeta_i^\mu - \sum_k w_{ik} \xi_k^\mu \right)^2 \quad (53)$$

The smaller the value is, the better the perceptron behaves; when it becomes zero, the matching condition (38) is naturally fulfilled.

Note the cost function depends on the connection strength matrix \mathbf{w} and the input patterns ξ^μ , whereas the Hamilton (14) depended on the current state \mathbf{S} , which evolved to minimise the energy, but \mathbf{w} is a constant. Here, \mathbf{w} is updated through the learning process.

A common algorithm for training the perceptron is the gradient descent learning:

$$\Delta w_{ik} = -\eta \frac{\partial E}{\partial w_{ik}} = \eta \sum_{\mu} (\zeta_i^\mu - O_i^\mu) \quad (54)$$

The name follows from the optimisation technique which tries to find the minimum along the gradient. If the changes is made for every input pattern in turn, the rule is reduced into

$$\Delta w_{ik} = \eta \delta_i^\mu \xi_k^\mu \quad (55)$$

where δ_i^μ is the errors (also known as deltas) given by

$$\delta_i^\mu = \zeta_i^\mu - O_i^\mu \quad (56)$$

This algorithm is often called the delta rule, or the LMS (least mean square), or the adaline rule, or the Widrow-Hoff rule [24, 25], which enables the network to find the optimal connection by successive improvement.

Certainly, the existence condition for such optimal \mathbf{w} is still the linear independence suggested by Equation (50). With the minima exists, the updating process shall make the network converges in an asymptotic way (for the proof see [2]).

Note Equation (55) is in exactly the same form as Equation (43) for the threshold neurons. Equation (43) was an educated guess based on the Hebb rule but Equation (55) here is naturally derived with gradient descent method. The advantage of this new approach will emerge when multi-layer network is studied, because it is more easily generalised than the Hebb rule in such cases.

Despite of the learning rule in the same form, threshold perceptrons have many differences from linear ones. First of all, the neurons in the two models takes different values, binary versus continuous. The second difference is the conditions for existence of a solution, linear separability versus linear independence. Finally, networks with threshold neurons have their connection strengths converge in finite steps but linear networks approach the optimal perceptron asymptotically.

3.1.3 Nonlinear Neurons

Perceptrons with nonlinear neurons generally have no explicit solutions as the linear networks do. Nonetheless, the gradient descent learning technique could be easily generalised to any differentiable $g(h)$. The cost function now becomes

$$E(\mathbf{w}) = \frac{1}{2} \sum_{i\mu} (\zeta_i^\mu - O_i^\mu)^2 = \frac{1}{2} \sum_{i\mu} \left[\zeta_i^\mu - g\left(\sum_k w_{ik} \xi_k^\mu\right) \right]^2 \quad (57)$$

with the gradient being

$$\frac{\partial E}{\partial w_{ik}} = - \sum_{\mu} [\zeta_i^\mu - g(h_i^\mu)] g'(h_i^\mu) \xi_k^\mu \quad (58)$$

Hence the update of the connection strengths could be written in the same form as Equation (55) as the product of the input and the error, scaled by the learning rate, but now the delta is different from Equation (56):

$$\delta_i^\mu = (\zeta_i^\mu - O_i^\mu) g'(h_i^\mu) \quad (59)$$

as there is an additional factor of $g'(h_i^\mu)$ that is the derivative of the activation function. Similar to the form of Equation (18), the choice of $g(h)$ is often sigmoid, which could be still explained as a stochastic activation (see Section 3.1.4), but here the neurons are modelled as able to active at continuously different levels (for values from -1 to 1 representing 'very low activity' to 'very high activity').

A particular convenient and useful choice is the hyperbolic tangent function $g(h) = \tanh(\beta h)$, whose derivative is $g'(h) = \beta(1 - g^2)$. This derivative is large when $|h_i^\mu|$ is small, so at the early stage of learning (when \mathbf{w} is typically small), the update of \mathbf{w} is very effective, while $|\mathbf{w}|$ grows throughout learning and thus $|h_i^\mu|$ decays which makes the updates more cautious.

Another good property of the sigmoid activation function is that it makes the condition for the existence of a solution the same as for the linear perceptron, that is, linear independence between input patterns. Since the function is invertible, to solve the present problem with the desired patterns ζ_i^μ is equivalent to solve the linear one with $g^{-1}(\zeta_i^\mu)$.

However, the gradient descent may run into a local minima with nonlinear neurons, rather than the global one.

Another cost function, known as the relative entropy, is of common interest [26, 27, 28]

$$E = \sum_{i\mu} \left[\frac{1}{2} (1 + \zeta_i^\mu) \log \frac{1 + \zeta_i^\mu}{1 + O_i^\mu} + \frac{1}{2} (1 - \zeta_i^\mu) \log \frac{1 - \zeta_i^\mu}{1 - O_i^\mu} \right] \quad (60)$$

because it gives the delta

$$\delta_i^\mu = \beta(\zeta_i^\mu - O_i^\mu) \quad (61)$$

which effectively omits the derivative factor in Equation (59) and is the same as Equation (56) in the linear network and Equation (43) in the threshold case.

Since any differentiable function of ζ_i^μ and O_i^μ with minimum at $\zeta_i^\mu = O_i^\mu$ could be applied with the gradient descent learning, it is possible in practice to find a suitable form of delta first and construct cost function accordingly to acquire computational or analytical convenience.

3.1.4 Stochastic Neurons

Another generalisation is to use binary neurons S_i^μ with stochastic thresholds defined by Equation (18):

$$\Pr(S_i^\mu = \pm 1) = \frac{1}{1 + \exp(\mp 2\beta h_i^\mu)} \quad (62)$$

with the net input $h_i^\mu = \sum_k w_{ik} \xi_k^\mu$ as usual. This gives, exactly as Equation (21),

$$\langle S_i^\mu \rangle = \tanh \left(\beta \sum_k w_{ik} \xi_k^\mu \right) \quad (63)$$

Then $\langle S_i^\mu \rangle$ plays the role of O_i^μ in contribution to the delta

$$\delta_i^\mu = \zeta_i^\mu - \langle S_i^\mu \rangle \quad (64)$$

which is simply the average over individual possible updates using the linear delta rule (56).

Before closing this section, it might be interesting to mention the memory capacity as for the Hopfield model. For perceptrons with continuous neurons, linear independence condition directly implies the answer: $p_{\max} = N$. It is more difficult to solve the problem for threshold cases. For the networks with random continuous inputs, Cover derived the solution back in 1965 [29], which is remarkably simple: $p_{\max} = 2N$. This result becomes exact in the large N limit but the number of the output neurons must be fixed.

Simple perceptrons of various types have been studied. Although it is hardly possible to find a real analogy in the nervous system, it has been shown even with such simple structures, the networks are capable of learning some computational tasks. The core technique of training, the gradient descent learning, will be directly adopted to multi-layer networks.

3.2 Multi-Layer Networks

Multi-layer perceptrons, by definition, have one or more hidden layers. They have a more realistic architecture comparing with simple perceptrons, although only those of the strictly feed-forward structure are to be studied, while natural neural networks normally have loops, direct or indirect.

The desired computation is the same as before, to match the input pattern with a desired one, i.e. Equation (38). As one may expect, they are more powerful, being able to overcome the limitation of simple perceptrons. A network with only one intermediate layer can calculate any Boolean function. For example, Figure 14 gives a solution to the XOR problem. The numbers

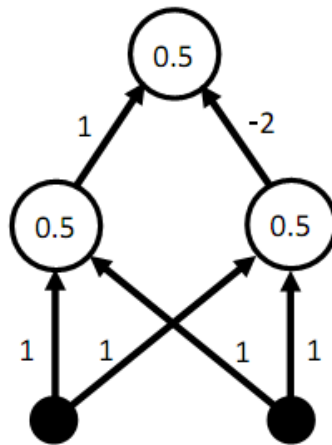


Figure 14: A two-layer perceptron solving the XOR problem: the neurons are binary taking 0 or 1 with the same threshold 0.5; the numbers on the arrows is the connection strengths.

in the neurons (as circles) are the thresholds and those on the connections (as arrows) specify the strengths. Note the hidden neurons on the left computes the logical OR, that on the right operates AND (right) and the output neuron computes OR again.

The simple example apparently follows from basic logical gate in electronics or the simple learning algorithm for threshold neurons in Section 3.1.1, but to evaluate the connection strengths in a multi-layer perceptron, the gradient descent learning introduced in Section 3.1.2 is applied. However, before that, the architecture of the network has to be designed by hand, that is, the number of hidden layers with neurons locating on each of them must be predetermined. For particular applications, educated guesses or effortful trials are of common use, while there are several instructive approaches to perform this task, though not constructive.

The particular supervised learning technique to be applied is called back-propagation, which lies

at the very centre of the study of the multi-layer perceptrons in this dissertation.

3.2.1 Scheme of Back-Propagation

Before the back-propagation algorithm was invented, there had been no learning rules for multi-layer perceptrons, which caused the interest in layered networks waning until late 1980s [2].

Being a learning rule, the algorithm specifies the form of Δw_{pq} as usual, depending on the training set (ξ_k^μ, ζ_k^μ) . It is based on gradient descent, the same as in Section 3.1.2 and 3.1.3. The core idea is to propagate errors backwards for readjustment while information flows along the feed-forward network.

It is not realistic to think this algorithm also applies to real neural networks; back-propagation is not directly linked to the Hebb rule. However, it might be still reasonable to consider similarities in the metaphysical sense. Conceptually, the model mimics a top-down network, which sends feedbacks to neurons at the earlier stage along its information process pathway. Although the mechanisms could be completely different, effective feedbacks must be sent because otherwise the neurons have nothing to adapt from.

For notatioanl convenience, two-layer perceptrons (see Figure 15) are mainly considered. Input

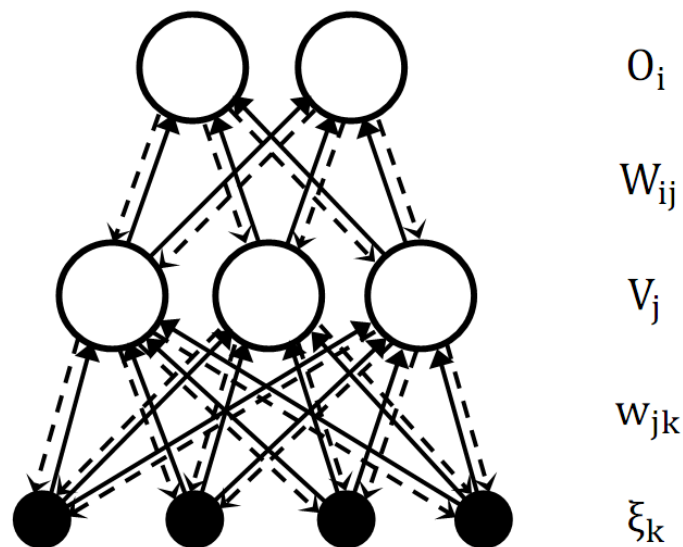


Figure 15: A two-layer perceptron with the notations for neurons and connections: the solid arrows are the flow of information and the dotted ones represents the back-propagation of errors.

terminals are denoted by ξ_k , outputs by O_i and hidden neurons by V_j . The connections from

the input neurons to the hidden ones are presented by w_{jk} and those from the hidden ones to the outputs by W_{ij} . Note the index k always refers to the input neurons, j to the hidden ones and i to outputs.

Signals are propagating forwards in the direction the solid arrows as shown in Figure 15. Starting from pattern μ , hidden neuron j receives a net input,

$$h_j^\mu = \sum_k w_{jk} \xi_k^\mu \quad (65)$$

and computes its output

$$V_j^\mu = g(h_j^\mu) \quad (66)$$

Then output neuron i receives

$$h_i^\mu = \sum_j W_{ij} V_j^\mu \quad (67)$$

and produces the final output

$$O_i^\mu = g(h_i^\mu) = g\left(\sum_j W_{ij} g\left(\sum_k w_{jk} \xi_k^\mu\right)\right) \quad (68)$$

Note the inputs might be still binary but since g is continuous, the hidden and output neurons are always evaluated continuously.

The usual cost function (53) can now be written as

$$E[\mathbf{w}] = \frac{1}{2} \sum_{\mu i} \left[\zeta_i^\mu - g\left(\sum_j W_{ij} g\left(\sum_k w_{jk} \xi_k^\mu\right)\right) \right]^2 \quad (69)$$

This is clearly differentiable for every connection strength, so the gradient descent learning could be applied.

For the hidden-to-output connections,

$$\Delta W_{ij} = -\eta \frac{\partial E}{\partial W_{ij}} = \eta \sum_{\mu} \delta_i^\mu V_j^\mu \quad (70)$$

where the error,

$$\delta_i^\mu = g'(h_i^\mu) [\zeta_i^\mu - O_i^\mu] \quad (71)$$

has the same form as Equation (59) for non-linear simple perceptrons.

For the input-to-hidden connections, using the chain rule to differentiate E ,

$$\begin{aligned}
\Delta w_{jk} &= -\eta \frac{\partial E}{\partial w_{jk}} = -\eta \sum_{\mu} \frac{\partial E}{\partial V_j^{\mu}} \frac{\partial V_j^{\mu}}{\partial w_{jk}} \\
&= \eta \sum_{\mu} [\zeta_i^{\mu} - O_i^{\mu}] g'(h_i^{\mu}) W_{ij} g'(h_j^{\mu}) \xi_k^{\mu} \\
&= \eta \sum_{\mu} \delta_i^{\mu} W_{ij} g'(h_j^{\mu}) \xi_k^{\mu} \\
&= \eta \sum_{\mu} \delta_j^{\mu} \xi_k^{\mu}
\end{aligned} \tag{72}$$

with

$$\delta_j^{\mu} = g'(h_j^{\mu}) \sum_i W_{ij} \delta_i^{\mu} \tag{73}$$

Note ΔW_{ij} and Δw_{jk} are presented in the same form. In general, this form for back-propagation could be generalised to any multi-layer perceptrons:

$$\Delta w_{pq} = \eta \sum_{\text{patterns}} \delta_{\text{output}} \times V_{\text{input}} \tag{74}$$

which only considers the ends of the particular connection concerned, with neuron q as the input-end feeding forwards signal V and neuron p as the output-end contributing error δ . For the final layer of connections, the deltas are given by Equation (71), while they are in the form of Equation (73) for all other layers. This generalised result is derived simply by further differentiation.

Note the errors are propagated backwards with the 'forward' connection strength matrix. This might be the most different point from a real neural network. A natural nervous system of feed-forward and top-down synapses must have different routes for information flowing in and feedbacks out; it is nearly impossible for neuron A to receive feedbacks from neuron B given A's dendrites have no contact with B's axons.

Applying the learning rules (70) and (72) literally to update connections after summing over all patterns, known as the batch mode, is not computationally economic. Usually, the connections readjust themselves for each presented pattern incrementally. Clearly this approach would not affect the solution and it is even better when the training pattern is chosen randomly, because the system would explore the cost surface stochastically and thus widely, reducing the risk of being captured by local minima.

There are two important consequences led by the learning rule (74), which makes the back-propagation algorithm particularly powerful in practice [2]:

1. The rule is local. Only quantities from two ends of the connection concerned are necessary to compute the corresponding update. This enables parallel computation as real nervous systems.
2. The algorithm is computationally economic. If there are n connections in the network, to compute the cost function takes operations of order n , so directly calculating n derivatives requires operations of order n^2 . In contrast all the derivatives are to be calculated in operations of order n with the back-propagation algorithm.

As before, the usual choice of the sigmoid function $g(h)$ is

$$g(h) = f_\beta(h) = \frac{1}{1 + \exp(-2\beta h)} \quad (75)$$

for networks with neurons ranging from 0 to 1, or

$$g(h) = \tanh \beta h \quad (76)$$

for cases with neural activation of interval $[-1, 1]$. They are convenient because their derivatives can be expressed by the functions themselves; no additional computation is required. For Equation (75), $g'(h) = 2\beta g(1 - g)$ and for Equation (76, $g'(h) = \beta(1 - g^2)$). Therefore the form of deltas in the last layer is considerably simplified (which could be very complicated due to bad choice of $g(h)$). For example, Equation (71) can be easily presented as

$$\delta_i^\mu = O_i^\mu(1 - O_i^\mu)(\zeta_i^\mu - O_i^\mu) \quad (77)$$

with $\beta = 1$ for 0/1 neurons.

3.2.2 Alternatives for Cost Function

As aforementioned, any differentiable E could be chosen as the cost function as long as it is minimised when $\zeta_i^\mu = O_i^\mu$. It is straightforward to show that only deltas of the output layer, that is, in the form of Equation (71) would change accordingly; all other deltas stay the same as in Equation (74).

The entropic measure (60) from Section 3.1.3 appears to a particularly good choice. It is for ± 1 outputs and, using $g(h) = \tanh h$, the error becomes

$$\delta_i^\mu = \zeta_i^\mu - O_i^\mu \quad (78)$$

Note the factor of $g'(h)$ disappears (for the output layer only) comparing to Equation (71). This accelerates learning when $|h|$ is large as $g'(h)$ is small. However, it could cause oscillation or

overshoot with h near 0.

It was found that the compromise of Equation (71) and (78) works better than both of them [30]:

$$\delta_i^\mu = [g'(h_i^\mu) + 0.1](\zeta_i^\mu - O_i^\mu) \quad (79)$$

It keeps $g'(h)$ but ensures δ_i^μ non-zero for large $|h|$, so less oscillations or overshoots will occur.

A different approach to make improvement is to modify the factor of $\zeta_i^\mu - O_i^\mu$. For instance,

$$\delta_i^\mu = \operatorname{arctanh} \frac{1}{2}(\zeta_i^\mu - O_i^\mu) \quad (80)$$

which enables the network learns more from greater errors (δ_i^μ grows faster than the increase of $|\zeta_i^\mu - O_i^\mu|$) [30].

As pointed out in Section 3.1.3, it might be more convenient to build the appropriate delta at the first place and thus construct the cost function. In fact, for purely computational aims, it might be unnecessary to know the explicit cost function, as the network computation is performed locally and parallel, while cost function is a global measure of the system.

Another approach is to modify the cost function through learning. The idea is to smooth out the cost surface at first to allow the system reach the local area where the global minimum locates and eventually depict the detailed landscape of the cost space to make the global minimum reveal itself. An explicit example is given by [31]

$$E = \begin{cases} \sum_{\mu i} (\zeta_i^\mu - O_i^\mu)^2 \gamma & \text{if } \operatorname{sgn}(\zeta_i^\mu O_i^\mu) = 1 \\ \sum_{\mu i} (\zeta_i^\mu - O_i^\mu)^2 & \text{if } \operatorname{sgn}(\zeta_i^\mu O_i^\mu) = -1 \end{cases} \quad (81)$$

with γ grows from 0 to 1 gradually. This algorithm makes the network focus on learning the correct sign first and then start to adapt the magnitude.

Certainly there are many other alternatives to the cost function and these different ideas to improve learning could be combined to become more powerful in practice. It will be interesting to study which cost function best models real networks and clearly this should be highly dependent on the particular part of nervous system being concerned.

3.2.3 Extensions of Learning Rate

The learning rate η directly scales deltas and thus determines how fast the network learns. As mentioned in Section 3.1.1, an appropriate value for η is important: if small, the connection

strengths may converge too slow; if large, oscillation may occurs. A simple but effective solution to the problem is the construction of momentum [32], which has become commonly applied.

The idea is to impose some inertia or momentum on w_{pq} , so that it explores the cost surface with newly presented information as a 'force' changing the tendency of changes, instead of deciding it directly. This can be realised by

$$\Delta w_{pq}(t+1) = -\eta \frac{\partial E}{\partial w_{pq}} + \alpha \Delta w_{pq} \quad (82)$$

where α , called the momentum parameter, has to be between 0 and 1; a common choice is 0.9. Note Δw_{pq} becomes a function of time as well, but the network computation could still be performed essentially at local neurons, because the rule is defined recursively and only information from the last learning epoch is required.

If a flat region of the cost surface is being explored, $\partial E / \partial w_{pq}$ is approximately the same at every time step. Hence, Equation (82) converges to

$$\Delta \approx -\frac{\eta}{1-\alpha} \frac{\partial E}{\partial w_{pq}} \quad (83)$$

with the effective learning rate being $\eta / (1 - \alpha)$, while in an oscillatory situation, it is still η . The system thus slips into the plateau of the cost surface faster, where the global minimum locates. Nonetheless, it is difficult to choose an appropriate value for η . Moreover, an optimal choice at the beginning may become not so good in later stages of learning. A useful technique is to adjust the value automatically during the procedure [33].

The idea is to check whether or not the current Δw_{pq} does actually reduce the cost. If not, the system overshoots, and thus η should be adjusted smaller. On the other hand if several successive updates have kept decreasing the cost, a larger η ought to be attempted. It seems a clever trick to apply

$$\Delta \eta = \begin{cases} +a & \text{if } \Delta E < 0 \text{ consistently;} \\ -b\eta & \text{if } \Delta E > 0; \\ 0 & \text{otherwise} \end{cases} \quad (84)$$

where ΔE is the cost function change due to Δw_{pq} , and a and b constants, so that several 'good' steps increase η linearly but a 'bad' step will lead to a geometric decay in it. This method is trying to accelerate learning without amplifying oscillations. The meaning of 'consistently' is to be decided in more detailed models in practice.

Although this adaptation of learning rate is at the expense of computational storage and complexity (as an individual neuron has to memorise a period of history or even perform some

certain averaging work over it), it is worthwhile in the sense not only the problem to choose an optimal value for η is solved, but also the computation is still taken place parallel at all neurons. A further generalisation is inspired that each connection has its own learning rate η_{pq} [33]. The learning performance could be better even without an adaptive rule like Equation (84). $\eta_{pq} \propto 1/(\text{fan-in of site } i)$ is a common choice [32].

The back-propagation algorithm give a general architecture in perceptrons performing the learning task. Optimisation techniques other than gradient descent could be used in the algorithm. For example, the steepest descent method [34, 35], which is based on line searches, conjugate gradient methods [31, 35, 36], which combines gradient information in successive steps, and quasi-Newton method [34, 36], which exploits the Newton rule for optimisation but avoids the computation of the inverse Hessian of the cost function. These methods still require information of the first derivatives only, so they fits the back-propagation algorithm well; network computation is performed by individual neurons locally.

3.3 Simulation and Discussion

3.3.1 Algorithm Review

Section 3.2.1 offers a comprehensive introduction and analysis of the back-propagation algorithm. Since it is important, the algorithm is summarised here. In addition, technical details concerning simulations will be adapted.

The network is made up of M layers denoted by $m = 1, 2, \dots, M$. V_i^m represents the output of the neuron i in the layer m . w_{ij}^m is the connection strength from V_j^{m-1} to V_j^m . Note the superscript m here is of the index of some layer, instead of μ for different patterns, which is saved because here w_{ij} is updated incrementally; distinguishing μ is of no help.

With the above set up, the procedure is:

1. Initialise the connection strengths. As the starting point on the cost surface, the initial value of w_{ij} is important. If its size is too large, the sigmoid activation function will saturate too early, and thus the system is captured by a local minimum near the starting point. A prescription is to make the size comparable with the typical net input, which could be obtained by making w_{ij} of the order of $1/\sqrt{k_i}$ where k_i is the number of pre-synaptic neurons of neuron i [2].

2. Choose a pattern ξ_k^μ randomly and feed it to the input terminals ($m = 0$), that is, for all k ,

$$V_k^0 = \xi_k^\mu \quad (85)$$

3. Propagate forwards the signal from layer $m - 1$ to m successively by

$$V_i^m = g(h_i^m) = g\left(\sum_j w_{ij}^m V_j^{m-1}\right) \quad (86)$$

for all i and each m until V_i^M , the final outputs, have been computed.

4. Compare the outputs with the desired match ζ_i^μ and calculate the errors:

$$\delta_i^M = g'(h_i^M)[\zeta_i^\mu - V_i^M] \quad (87)$$

5. Propagate backwards the errors from layer m to $m - 1$ successively by

$$\delta_i^{m-1} = g'(h_i^{m-1}) \sum_j w_{ji}^m \delta_j^m \quad (88)$$

for all i and each m until every neuron 'knows' its error.

6. Adjust all connections by

$$\Delta w_{ij}^m = \eta \delta_i^m V_j^{m-1} \quad (89)$$

7. Go back to step 2 and repeat to learn the next pattern, until the end of learning. Note the stopping time is to be determined, or based on some criteria that, when the system meets, the experimentalist is satisfied.

3.3.2 Overfitting Problem

Overfitting is often met in statistics and machine learning. A naive example is to fit a set of points lying in the neighbourhood of a line with polynomials of high orders. The curve might be perfect to trace the points, but it hardly yields good predictions of new cases. Since there are too many free parameters, the model tries to represent the noise, rather than the regularity of the data.

For neural networks, it usually has a consider number of free parameters of thresholds and connections; for a training process, there could be even more, such as learning rates. Additionally,

the realistic problem to solve is commonly beyond the human intuition; not like points on a 2-D plane, one can feel whether they lie along a straight line. Therefore, it is often difficult to construct a network of the optimal size to avoid overfitting. It is well studied both in theory [37] and practice [38] that neural networks with too many parameters do overfit and thus are unable to generalise well to new situations. Hence, there were many efforts to simplify the networks, such as removing the connection weights [39].

Another common method to prevent overfitting is using a validation set, also called self-test set. Normally data is divided into training set, which is to be learnt by the model, and test set, which is used to test whether the model well generalises. A validation set is no more than another test set but keeping the model operating self tests during learning. The error against the validation set normally decreases alongside that against the training set but then increases when overfitting occurs, while the training set error would continue its decay.

This approach is based on the empirical evidence that shows large networks are not likely to overfit than expected if they could stop learning at a relatively early stage. Caruana et al. (2001) [40] explains that a large network firstly learns a simple model similar to the one learnt by a simpler model of the optimal size, so the model would generalise well if stops learning at this appropriate time.

3.3.3 Toy Example

A naive example is to be simulated with MATLAB to examine the various alternatives of perceptrons that have been studied. Suppose a common fair die is thrown for six times and the results are recorded: ξ_1 for the first time, ξ_2 for the second, etc., and the vector $\xi^\mu = (\xi_k^\mu)_{k=1,2,3,4}$ is just an input pattern for the perceptron. Due to the simplicity of the example, there are at most $46656 (= 6^6)$ different patterns. (In contrast, pattern spaces of realistic problems are much larger; even as simple as in Section 2.3, it has the cardinality of $2^{30 \times 30} > 10^{270}$). Three features, denoted by ζ^μ , could be obtained from any pattern ξ^μ :

1. whether the sum of the six trials is odd ($\zeta_1^\mu = 1$) or even (-1);
2. whether the sum of the six trials is less than 20 ($\zeta_2^\mu = 1$), or not (-1); and
3. whether the sum of the first three trials is less than that of the rest ($\zeta_3^\mu = 1$), or not (-1).

Hence, the desired network computation of the perceptron is to match the pair (ξ^μ, ζ^μ) , that is, to fulfil the association condition (38).

ξ^μ is randomly generated and then corresponding ζ^μ is calculated explicitly. Repeating this procedure, data for the perceptron to learn and test could be easily prepared (see Code P1 in Appendix B.2). Note the activation function of choice is $g(h) = \tanh(h)$, which allows neurons to take continuous values. The learning algorithm has been summarised in Section 3.3.1 and each entire learning process will be repeated for 100 times to average the effect of random patterns (see Code P2 and P3 in Appendix B.2). The mean behaviour of the error on the learning set and the test set will be a measure of the network performance as a memory system.

Note the random generated learning and test sets can and are likely to, when large, have repeated patterns. This should not be a problem because the probability of having many same patterns is very low, and even in the extreme situation, these same patterns would not affect the minima (either local or global).

Simulation 1: simple perceptrons

To realise this easy network computation, a simple perceptron might be good enough. Its structure is completely determined by the inputs and outputs, but since the thresholds are absorbed by ξ_0 , there is virtually 7(= 6 + 1) neurons as inputs.

The four choices of deltas, Equation (77), (78), (77) and (80), are compared in Figure 16. Each trajectory shows error percentage on the training set itself or the test set against learning time, i.e. number of patterns learnt. Note for every delta, there are three cases of different learning rates: small ($\eta = 0.1$), medium (1) and large (10). Error percentages for different features ζ_i are not summarised but presented separately. Hence, there are many paths, but since only the deltas are being concerned, all sub-cases of one delta are in the same colour.

It is reasonable to ignore those details as the error behaviour is generally determined by the choice of delta (cost function). The blue and red paths are for deltas $\delta_i^\mu = O_i^\mu(1 - O_i^\mu)(\zeta_i^\mu - O_i^\mu)$ and $\delta_i^\mu = \zeta_i^\mu - O_i^\mu$ respectively. They are apparently not so good as $\delta_i^\mu = [O_i^\mu(1 - O_i^\mu) + 0.1](\zeta_i^\mu - O_i^\mu)$ (green) and $\delta_i^\mu = \text{arctanh}(\zeta_i^\mu - O_i^\mu)/2$ (cyan), as the green and cyan paths have reached 0. The networks with such deltas may converge too slow, or in an even worse situation, they are converging to a wrong solution. Nonetheless, the red oscillations, caused by the entropic delta, are remarkable (recall each path averages 100 trials) just as the theory [30] suggests.

Now that it is known from both theory and practice $\delta_i^\mu = [O_i^\mu(1 - O_i^\mu) + 0.1](\zeta_i^\mu - O_i^\mu)$ and $\delta_i^\mu = \text{arctanh}(\zeta_i^\mu - O_i^\mu)/2$ are better, their details should be examined. Before that, a mean error of those on the three features (equally weighted) is introduced to make the graph less messy.

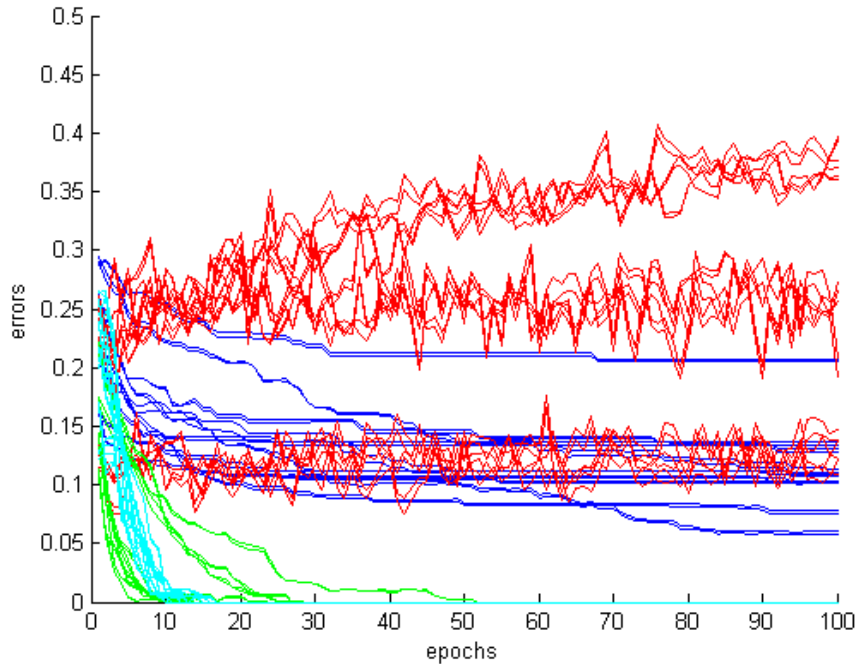


Figure 16: Errors on the training and test sets of the simple perceptron with the different deltas: the blue trajectories are for $\delta_i^\mu = O_i^\mu(1 - O_i^\mu)(\zeta_i^\mu - O_i^\mu)$, the red for $\delta_i^\mu = \zeta_i^\mu - O_i^\mu$, the green for $\delta_i^\mu = [O_i^\mu(1 - O_i^\mu) + 0.1](\zeta_i^\mu - O_i^\mu)$ and the cyan for $\delta_i^\mu = \text{arctanh}(\zeta_i^\mu - O_i^\mu)/2$.

Since it could be noted that the variances the error behaviour are mainly decided by the deltas, the averages are representative.

Figure 17 zooms into the early stage of learning. The trajectories with dots are for $\delta_i^\mu = \text{arctanh}(\zeta_i^\mu - O_i^\mu)/2$. The blue ones represents $\eta = 0.1$ with both the deltas, red for $\eta = 1$ and green for $\eta = 10$.

Note there are almost no differences between the errors on the training set and those on the test set. This might be caused by random pattern in both the sets, or it is just because this example is too easy. In either the case, it means the simple perceptrons could be generalised very well to new cases (since they are likely to meet new cases in the test set). The networks are very powerful in performing the desired computation with no more than 25 patterns learnt.

The learning rate η does determine the speed of convergence for $\delta_i^\mu = [O_i^\mu(1 - O_i^\mu) + 0.1](\zeta_i^\mu - O_i^\mu)$, but it is interesting that for $\delta_i^\mu = \text{arctanh}(\zeta_i^\mu - O_i^\mu)/2$ the medium rate $\eta = 1$ results in faster learning. Recall all the paths averages trial of 100 times, which apparently erases the oscillation suggested by the theory, but it could be claimed for this example that the medium learning rate is the best on average among the three values; it is not even boSelf and Test Errorsthering to discuss whether the possibility to find the optimal connections faster is worthy of the risk of

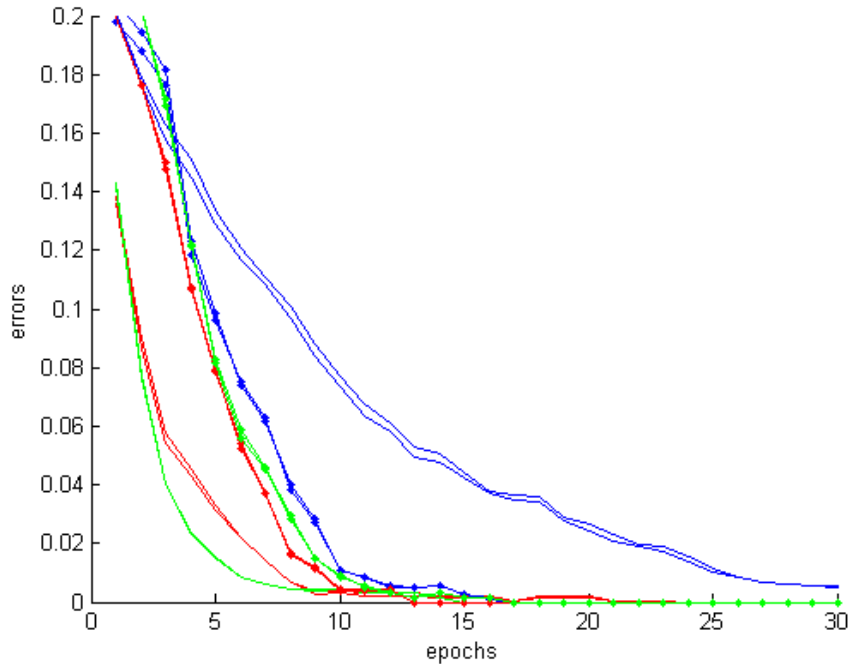


Figure 17: Errors on the training and test sets of the simple perceptron: the smooth curves are for $\delta_i^\mu = [O_i^\mu(1 - O_i^\mu) + 0.1](\zeta_i^\mu - O_i^\mu)$ and those with dots represents $\delta_i^\mu = \text{arctanh}(\zeta_i^\mu - O_i^\mu)/2$; all blue paths are for $\eta = 0.1$, red for $\eta = 1$ and green for $\eta = 10$.

wandering into other attractors.

Considering various performances of η , it appears that the best choice of delta is $\delta_i^\mu = \text{arctanh}(\zeta_i^\mu - O_i^\mu)/2$. It actually reduces the effect of the learning rate; in some sense, it has already made η adaptive. It is quite a pity that the extensions of the learning rate in Section 3.2.3 are not simulated and compared together, because the perceptron could learn so fast that very few effective updates of η would occur before the end of training.

Two-Layer Perceptron:

The simple perceptron has been quite a success; it might be hard for a two layer perceptron to outperform it. The network will have $7 = (6 + 1)$ input terminals and 3 output ones. Different number of hidden neurons will be checked; note for h intermediate neurons with thresholds, there shall be $h + 1$ ones without thresholds effectively (same as in the simple perceptron case). In order to be comparable with the one-layer case, let $\delta_i^\mu = \text{arctanh}(\zeta_i^\mu - O_i^\mu)/2$ and $\eta = 1$. In addition, blue paths are for the testing errors on the training set, red for those on the test set. Similarly, they are not expected to be noticeably different.

When the hidden layer consists of only one neuron (see Figure 18), The network learns slower

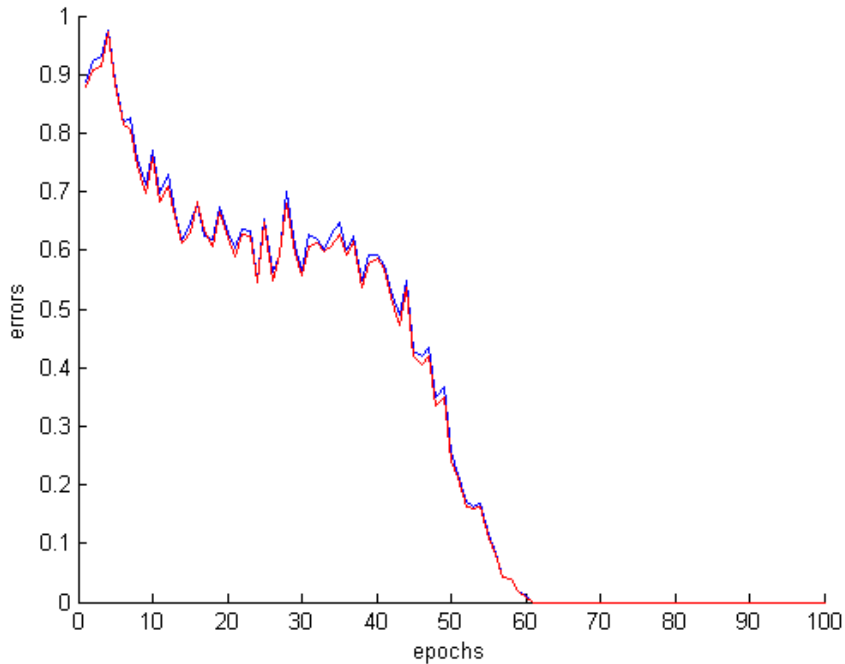


Figure 18: Errors on training and test sets of a 2-layer perceptron with 1 hidden neuron: the blue paths represents the errors on the training set and the red shows those on the test set.

than the simple perceptron, but its overall performance is still acceptable. It is not surprising that in order to estimate more parameters, more patterns are necessary to be learnt.

However, this expectation, or the opposite extreme case, too large networks may lead to overfitting, is totally mistaken. The results for the perceptrons with 2 to 9 hidden neurons are shown in Figure 19. Note the most green path is for 2 hidden neuron and the most red for 9; the performances become better successively with the increasing number of intermediate neurons, although the improvements are diminishing.

An extreme case of a two-layer perceptron with 100 hidden neurons is examined as well (see Figure 20). The results are remarkable; with only on average 6 patterns learnt, the network could perform the desired computation almost perfectly.

Note there are more than a thousand connections in this extremely large network. However, no overfitting problem ever occurs. The complication of the model is only improving the performances, which echoes the results of [40].

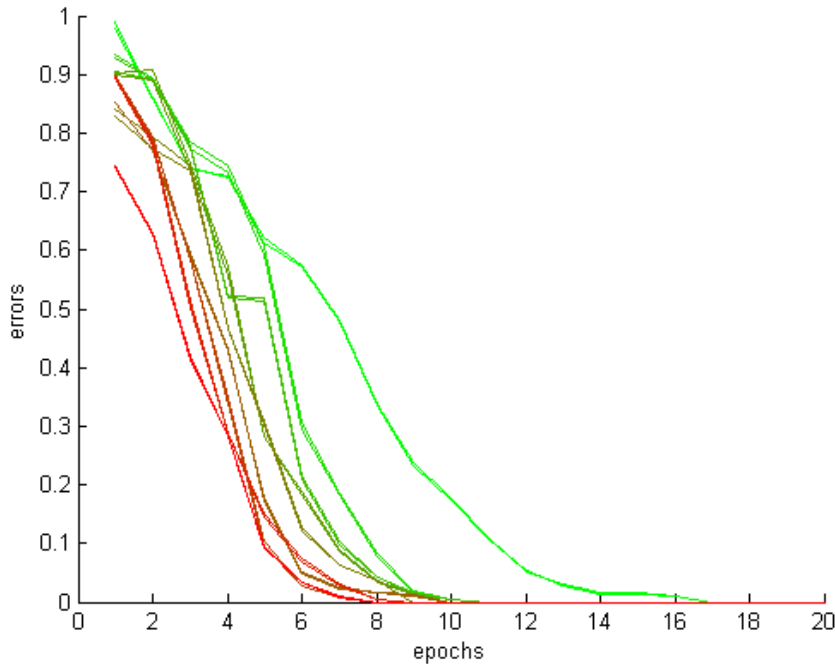


Figure 19: Errors on the training and test sets of a 2-layer perceptron: the most green curve is for the perceptron with 2 hidden neurons and the most red is for that with 9; the networks with 3-8 intermediate neurons are represented by the gradient colours in between.

3.4 Conclusion and Application

It has been shown that a perceptron, or a layered feed-forward network, is quite useful as a memory system and the additional interesting point is that the optimal connections are acquired by supervised learning. However, its link with natural neural networks is weak. One might argue the learning rules of the back-propagation algorithm are still generalisations of the Hebb rule, but it is not compatible with the natural neural networks, because the post-synaptic neuron cannot receive information from the pre-one (otherwise, the network becomes recurrent, rather a feed-forward).

Nonetheless, due to its effectiveness, perceptrons are widely used in realistic applications, such as the NETtalk project [41], trying to train a network to pronounce English, image compression [23] for high quality television signal transmission, and handwritten ZIP code recognition [38], helping the U.S. mail system to deal with the handwritten postal code.

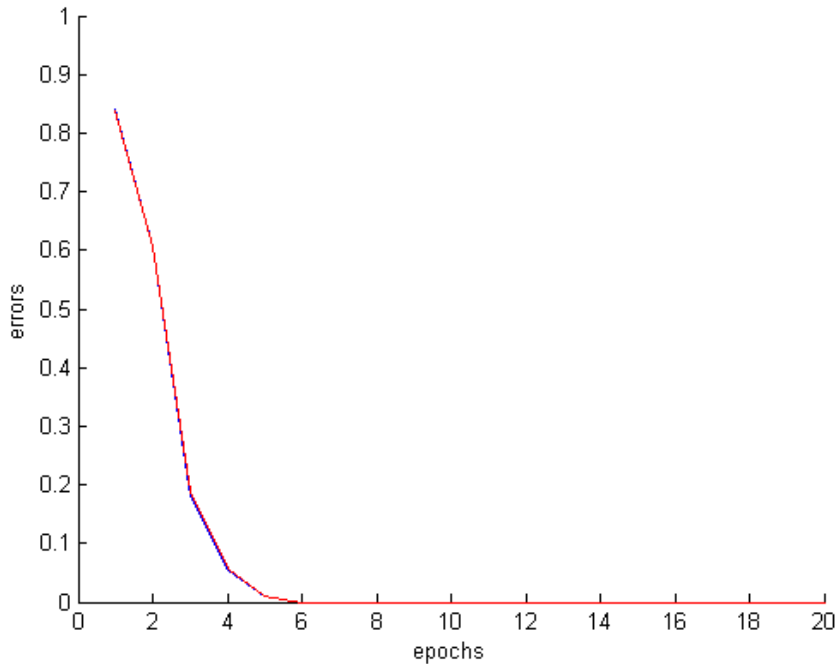


Figure 20: Errors on the training and test sets of a 2-layer perceptron with 100 hidden neurons: the blue paths represents the errors on the training set and the red shows those on the test set.

4 Conclusion

In this dissertation, two main models have been studied: the Hopfield model and the perceptron. They are both artificial neural networks capable of storing and extracting memories. They share the same fundamental assumption of McColluch-Pitts neurons, but have totally different structures. The most significant factor is the connection strengths in both the networks, where essentially the memories are stored.

In the Hopfield model, the networks are recurrent. The connection strengths are inspired and predetermined by the Hebb rule. A test input is then fed into the designed network and the most similar memory to the input is expected to emerge. Borrowing models from physics also brings in concepts such as energy and temperature, which plays an important role in analysing the memory capacity. The core object determines the capacity is called the crosstalk term, which is an interference produced by other memories when the system tries to recall a particular one. It has been shown that with some assumptions the capacity is quite large, but realistic problems may fail them. Further study may focus on the analysis of the structure of the crosstalk terms, which may offer an explanation for the unacceptable performance in the simulations and even a general solution.

The perceptron is a layered feed-forward network. It learns the optimal connections by supervised learning, particularly, the back-propagation algorithm. The initial inspiration is still the Hebb rule but gradient descent method appears to be more generalisable in multi-layer cases. The core idea of the algorithm is to update the connections through learning, which eventually reduces the cost function, built beforehand, measuring the performance of the network computation. The perceptron has been proved powerful in the desired task both in theory and in practice. However, a more realistic application should be investigated, in which it would be able to check whether the adaptive learning rate is a useful technique.

There are many other topics that have not been investigated by this dissertation. For instance, just pick one main feature from either the model has been studied, it would be interesting to understand how a recurrent network may learn. Actually, this was mentioned by the original paper of the Hopfield model [10], which found old memories are to be forgotten as soon as the newly added memory makes the system overloads. Additionally, Boltzmann machine has become popular and widely applied since introduced in 1980s [16]; it is particularly effective in finding the connection strengths for symmetric recurrent networks. Future works could be based on understanding the current theory and application of Boltzmann machine and heading in solving asymmetric recurrent networks.

References

- [1] Scoville W B and Milner B (1957). Loss of recent memory after bilateral hippocampal lesions. *Cognitive Neuroscience: A Reader*. 262-279. Oxford: Blackwell.
- [2] Hertz J, Krogh A and Palmer R G (1991). *Introduction to the theory of neural computation*. Redwood City: Addison-Weasley.
- [3] Schacter D L (1992). Understanding implicit memory: a cognitive neuroscience approach. *Cognitive Neuroscience: A Reader*. 305-324. Oxford: Blackwell.
- [4] Baddeley A, Eysenck M W and Anderson M (2009). *Memory*. New York: Psychology Press.
- [5] McCulloch W S and Pitts W (1943). A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*. **5**, 115-133
- [6] Dayan P and Abbott L F (2001). *Theoretical neuroscience*. 229-398. Cambridge: MIT Press.
- [7] Hebb D O (1949). *The organization of behavior: a neuropsychological theory*. New York: Wiley.
- [8] Koch C (2004). *Biophysics of computation: information processing in single neurons*. 308. Oxford: Oxford University Press
- [9] Marsland S (2009). *Machine learning : an algorithmic perspective*. Boca Raton: CRC Press.
- [10] Hopfield J J (1982). Neural networks and physical systems with emergent collective computational abilities. *Proceeding of the National Academy of Sciences, USA* **79**, 2554-2558.
- [11] Toulouse G, Dehaene S and Changeux J-P (1986). Spin glass model of learning by selection. *Proceeding of the National Academy of Sciences, USA*. **83**, 1695-1698.
- [12] Kanter I and Sompolinsky H (1987). Associative recall of memory without errors. *Physical Review A*. **35**, 380-392.
- [13] Amit D J, Gutfreund H and Sompolinsky H (1985) Spin-glass models of Neural Networks. *Physical Review A*. **32**, 1007-1018.
- [14] Amit D J, Gutfreund H and Sompolinsky H (1985) Storing infinite numbers of patterns in a spin-glass model of neural networks. *Physical Review Letter*. **55**, 1530-1533.

- [15] Glauber R J (1963). Time-dependent statistics of the Ising model. *Journal of Mathematical Physics*. **4**, 294-307.
- [16] Hinton G E and Sejnowski T J (1983). Optimal perceptual inference. In *Proceedings of the Eighth Annual Conference of the Cognitive Science Society* (Armherst 1986). 1-12. Hillsdale: Erlbaum.
- [17] Amit D J, Gutfreund H and Sompolinsky H (1987) Statistical mechanics of neural networks near saturation. *Annals of Physics*. **173**, 30-67.
- [18] Tsodyks M V and Feigel'man M V (1988). The enhanced storage capacity in neural networks with low activity level. *Europhysics Letters*. **6**, 101-105.
- [19] Lowe M (1998). On the storage capacity of hopfield models with correlated patterns. *The Annals of Applied Probability*. **8.4**, 1216-1250.
- [20] Rosenblatt F (1957). *The perceptron, a perceiving and recognizing automaton Project Para.* Cornell Aeronautical Laboratory.
- [21] Rosenblatt F (1957). *Principles of Neurodynamics*. New York: Spartan.
- [22] Bourland H and Kamp Y (1988). Auto-association by multilayer perceptrons and singular value decomposition. *Biological Cybernetics*. **59**, 291-294.
- [23] Cottrell G W, Munro P and Zipser D (1987). Learning internal representations from gray-scale images: an example of extensional programming. In *Ninth Annual Conference of the Cognitive Science Society* (Seattle 1987), 462-473. Hillsdale: Erlbaum.
- [24] Rumelhart D E, McClelland J L and the PDP Research Group (1986) *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Volume 1: Foundations*. Cambridge: MIT Press.
- [25] Widrow B and Hoff M E (1960) Adaptive switching circuits. In *1960 IRE WESCON Convention Record*, part 4, 96-104.
- [26] Baum E B and Wilczek F (1988). Supervised learning of probability distributions by neural networks. In *Neural Information Processing Systems* (Denver 1987). ed. Anderson D Z, 52-61. New York: American Institute of Physics.

- [27] Hopfield J J (1987) Learning algorithms and probability distributions in feed-forward and feed-back networks. *Proceeding of the National Academy of Sciences, USA* **84**, 8429-8433.
- [28] Solla S A, Levin E and Fleisher M (1988). Accelerated learning in layered neural networks. *Complex Systems*. **2**, 625-639.
- [29] Cover T M (1965). Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition. *IEEE Transactions on Electronic Computers*. **14**, 326-334.
- [30] Fahlman S E (1989) Fast-learning variations on back-propagation: an empirical study. In *Proceedings of the 1988 Connectionist Models Summer School* (Pittsburg 1988), eds. Tourestzky D, Hinton G and Sejnowski T, 38-51. San Mateo: Morgan Kaufmann.
- [31] Makram-Ebeid S, Sirat J-A and Viala J-R (1989). A rationalized back-propagation learning algorithm. In *International Joint Conference on Neural Networks* (Washington 1989). **2**, 373-380. New York: IEEE.
- [32] Plaut D, Nowlan S and Hinton G (1986). Experiments on learning by back-propagation. Technical Report CMU-CS-86-126, Department of Computer Science, Carnegie Mellon University, Pittsburgh, PA.
- [33] Jacobs R A (1988) Solution of the travelling salesman problem with an adaptive ring. In *IEEE International Conference on Neural Networks* (San Diego 1988). **1**, 85-92. New York: IEEE.
- [34] Watrous R L (1987). Learning algorithms for connectionist networks: applied gradient methods of nonlinear optimization. In *IEEE First International Conference on Neural Networks* (San Diego 1987), eds. Caudill M and Butler D. **2**, 619-627. New York: IEEE.
- [35] Kramer A H and Sangiovanni-Vincentelli A (1989). Efficient parallel learning algorithms for neural networks. In *Advances in Neural Information Processing System I* (Denver 1988). ed. Tourestzky D S, 40-48. San Mateo: Morgan Kaufmann.
- [36] Press W H, Flannery B P, Teukolsky S A and Vetterling W T (1986). *Numerical Recipes*. Cambridge: Cambridge University Press.
- [37] Baum E B and Haussler D (1989). What size net gives valid generalization? *Neural Computation*. **1**, 151-160.

- [38] Le Cun Y (1989). Generalization and network design strategies. *Connections in Perspective*. 143-55.
- [39] Le Cun Y, Denker J S and Solla S A (1990). Optimal brain damage. *NIPs*. **2**, 598-605.
- [40] Caruana R, Lawrence S and Giles L (2001). Overfitting in neural nets: backpropagation, conjugate gradient, and early stopping. *Advances in neural information processing systems*. 402-408.
- [41] Sejnowski T J and Rosenberg C R (1987). Parallel networks that learn to pronounce English text. *Complex Systems*. **1**, 145-168.

Appendices

A Large Picture



Figure 21: Results of the simulation with the large Chinese characters: there are five groups of results with each of 12 images exhibited in two lines; the first image of the first line in each group is the original test input and the rest are results after 1000, 2000, 3000, 4000, 5000, 10000, 20000, 30000, 40000 and 50000 times of updates.

B MATLAB Code

B.1 The Hopfield Networks

Code H1 reads the images and stores them in a 3-D matrix. No input other than the pictures is required.

```
nu=['1.bmp';'2.bmp';'3.bmp';'4.bmp';'5.bmp';'6.bmp';'7.bmp';'8.bmp';'9.bmp';'0.bmp'];
sl=['a.bmp';'b.bmp';'c.bmp';'d.bmp';'e.bmp';'f.bmp';'g.bmp';'h.bmp';'i.bmp';'j.bmp';
'k.bmp';'l.bmp';'m.bmp';'n.bmp';'o.bmp';'p.bmp';'q.bmp';'r.bmp';'s.bmp';'t.bmp';
'u.bmp';'v.bmp';'w.bmp';'x.bmp';'y.bmp';'z.bmp'];
bl=['bigA.bmp';'bigB.bmp';'bigC.bmp';'bigD.bmp';'bigE.bmp';'bigF.bmp';'bigG.bmp';
'bigH.bmp';'bigI.bmp';'bigJ.bmp';'bigK.bmp';'bigL.bmp';'bigM.bmp';'bigN.bmp';
'bigO.bmp';'bigP.bmp';'bigQ.bmp';'bigR.bmp';'bigS.bmp';'bigT.bmp';'bigU.bmp';
'bigV.bmp';'bigW.bmp';'bigX.bmp';'bigY.bmp';'bigZ.bmp'];
% 62 patterns, each with 30x30 pixels
pattern=zeros(30,30,62);
% pattern 1-10 are images of the 10 digits
for p=1:10
    picture=imread(nu(p,:));
    for i=1:30
        for j=1:30
            pattern(i,j,p)=picture(i,j);
        end
    end
end
% pattern 11-36 are images of the 26 letters in small case
for p=11:36
    picture=imread(sl(p-10,:));
    for i=1:30
        for j=1:30
            pattern(i,j,p)=picture(i,j);
        end
    end
end
% pattern 37-62 are images of the 26 letters in large case
for p=37:62
    picture=imread(bl(p-36,:));
    for i=1:30
```

```

        for j=1:30
            pattern(i,j,p)=picture(i,j);
        end
    end
end

```

Code H2 calculates the connection strength of the network. It requires the value of pattern produced by Code H1.

```

[h,l,p]=size(pattern);
N=h*l;
w=zeros(N,N);
xi=reshape(pattern,N,p)/255*2-1;
for mu=1:p
    xxi=xi(:,mu);
    w=w+xxi*xxi';
end
w=w/N;
for x=1:N
    w(x,x)=0;
end

```

Code H3 recalls the memory (the stored patterns) for a given input. It requires the value of connection strength produced by Code H2.

```

item1=imread('sample.bmp');
S=ones(h,l);
for x=1:h
    for y=1:l
        if item1(x,y)<=200
            S(x,y)=-1;
        else
            S(x,y)=1;
        end
    end
end
end

```



```

S=reshape(S,N,1);
for t=1:10000
    i=randi(N);
    S(i)=sign(w(i,:)*S);
end
item2=reshape(S,h,1);
imwrite(item2,'recall.bmp')

```

B.2 The Perceptrons

Code P1 randomly generates the results of throwing a die for six times and calculates the results of the three features explicitly.

```

function desired=aa0_sixdicegenerator(p)
xi=ones(7,p);
zeta=ones(1,p);
for mu=1:p
    dice=randi(6,6,1);
    fetr=ones(3,1);
    sumodc=sum(dice);
    sum123=sum(dice([1:3]));
    sum456=sumodc-sum123;
    if mod(sumodc,2)==0
        fetr(1)=-1;
    end
    if sumodc>=20
        fetr(2)=-1;
    end
    if sum123>=sum456
        fetr(3)=-1;
    end
    for k=1:6
        xi(k,mu)=dice(k);
    end
    xi(7,mu)=-1;
    for i=1:3

```

```

        zeta(i,mu)=fetr(i);
    end
end
desired=[xi;zeta];
end

```

Code P2 simulates a simple perceptron trying to learn the features (patterns) from the results.

```

clf
p=100;time=100;
eta=10;
avestem=0;avetem=0;
for t=1:time
stem=[];tem=[];
self_test_error=0;test_error=0;
xizeta=aa0_sixdicegenerator(p);xi=xizeta([1:7],:);zeta=xizeta([8:10],:);
testxizeta=aa0_sixdicegenerator(100);
testxi=testxizeta([1:7],:);testzeta=testxizeta([8:10],:);
O=zeros(3,1);w=zeros(3,7);
% ——— initialisation ———
for i=1:21
    w(i)=(1/sqrt(7)+randn)*(randi(2)*2-3);
end
% ——— learning ———
mord=randperm(p);
for mm=1:p
    mu=mord(mm);
    % ——— signal forwards ———
    hoO=w*xi(:,mu);
    for i=1:3
        O(i)=tanh(hoO(i));
    end
    % ——— error backwards ———
    doO=(zeta(:,mu)-O); % eqn.6.8/6.20/6.21
    % ——— connection update ———
    w=w+eta*doO*xi(:,mu)';
% ——— self test ———
V=zeros(3,p);

```

```

for mu=1:p
    hoV=w*xi(:,mu);
    for i=1:3
        V(i,mu)=(tanh(hoV(i)))*2-1;
    end
end
error=abs(V-zeta(:, [1:p]));
fserror=zeros(3,1);
for i=1:3
    fserror(i)=length(find(error(i,:)<1));
end
self_test_error=fserror/p;
% — test —
T=zeros(3,100);
for mu=1:100
    hoT=w*testxi(:,mu);
    for i=1:3
        T(i,mu)=(tanh(hoT(i)))*2-1;
    end
end
error=abs(T-testzeta);
fterror=zeros(3,1);
for i=1:3
    fterror(i)=length(find(error(i,:)<1));
end
test_error=fterror/100;
stem=[stem,self_test_error];
tem=[tem,test_error];
end
avestem=avestem+stem;
avetem=avetem+tem;
end
avestem=avestem/time;
avetem=avetem/time;
hold on
plot([10:10:p],avestem(:, [10:10:p]), 'b')
plot([10:10:p],avetem(:, [10:10:p]), 'r')

```

Code P3 simulates a two-layer perceptron trying to learn the features (patterns) from the

results.

```
clf
% There are 6+1 input terminals, 3 output units and one hidden layer with
% h+1 units.
p=100;time=100;
eta=1;h=100;
avestem=0;avetem=0;
for t=1:time
stem=[];tem=[];
self_test_error=0;test_error=0;
xizeta=aa0_sixdicegenerator(p);xi=xizeta([1:7],:);zeta=xizeta([8:10],:);
testxizeta=aa0_sixdicegenerator(100);
testxi=testxizeta([1:7],:);testzeta=testxizeta([8:10],:);
O=zeros(3,1);W=zeros(3,h+1);V=zeros(h+1,1);w=zeros(h+1,7);
% ——— initialisation ———
for i=1:(3*(h+1))
    W(i)=(1/sqrt(h+1)+randn)*(randi(2)*2-3);
end
for i=1:((h+1)*7)
    w(i)=(1/sqrt(7)+randn)*(randi(2)*2-3);
end
% ——— learning ———
mord=randperm(p);
for mm=1:p
    mu=mord(mm);
    % ——— signal forwards ———
    hoV=w*xi(:,mu);
    for j=1:(h+1)
        V(j)=tanh(hoV(j));
    end
    hoO=W*V;
    for i=1:3
        O(i)=tanh(hoO(i));
    end
    % ——— error backwards ———
    doO=atanh((zeta(:,mu)-O)/2);%
    doV=(1-V).*(W'*doO);
    % ——— connection update ———
```

```

W=W+eta*doO*V';
w=w+eta*doV*xi(:,mu)';
% — self test —
S=zeros(3,p);
for mu=1:p
    hotV=w*xi(:,mu);
    for j=1:(h+1)
        tV=(tanh(hotV(j)))*2-1;
    end
    hoS=W*tV;
    for i=1:3
        S(i,mu)=(tanh(hoS(i)))*2-1;
    end
end
error=abs(S-zeta(:, [1:p]));
fserror=zeros(3,1);
for i=1:3
    fserror(i)=length(find(error(i,:)<1));
end
self_test_error=sum(fserror)/p;
% — test —
T=zeros(3,100);
for mu=1:100
    hotV=w*testxi(:,mu);
    for j=1:(h+1)
        tV=(tanh(hotV(j)))*2-1;
    end
    hoT=W*tV;
    for i=1:3
        T(i,mu)=(tanh(hoT(i)))*2-1;
    end
end
error=abs(T-testzeta);
fterror=zeros(3,1);
for i=1:3
    fterror(i)=length(find(error(i,:)<1));
end
test_error=sum(fterror)/p;
stem(mm)=self_test_error;
tem(mm)=test_error;

```

```
end
avestem=avestem+stem;
avetem=avetem+tem;
end
avestem=avestem/time;
avetem=avetem/time;
hold on
plot([1:p],avestem(:, [1:p]), 'b')
plot([1:p],avetem(:, [1:p]), 'r')
```