Universal Approximation Theory of Neural Networks

by

Simon Odense
B.Sc., University of Victoria, 2013

A Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of

Master of Science

in the Department of Mathematics and Statistics

# Supervisory Committee

Universal Approximation Theory of Neural Networks

by

Simon Odense
B.Sc., University of Victoria, 2013

---

Dr. Roderick Edwards, Supervisor
(Department of Mathematics and Statistics)

---

Dr. Farouk Nathoo, Departmental Member
(Department of Mathematics and Statistics)

## ABSTRACT

**Supervisory Committee**

---

Dr. Roderick Edwards, Supervisor
(Department of Mathematics and Statistics)

---

Dr. Farouk Nathoo, Departmental Member
(Department of Mathematics and Statistics)

Historically, artificial neural networks have been loosely defined as biologically inspired computational models. When deciding what sort of network to use for a given task there are two things that need to be considered. The first is the representational power of the given network, that is what class of problems can be solved by this network? Given a set of problems to be solved by neural networks, a network that can solve any of these problems is called a universal approximator. The second problem is the ability to find a desired network given an initial network via a learning rule. Here we are interested in the question of universal approximation. A general definition of artificial neural networks is provided along with definitions for different kinds of universal approximation. We then prove that the recurrent temporal restricted Boltzmann machine (RTRBM) satisfies a general type of universal approximation for stochastic processes, an extention of previous results for the simple RBM. We conclude by examining the potential use of such temporal artificial neural networks in the biological process of perception.

# Table of Contents

# List of Figures

# ACKNOWLEDGEMENTS

I would like to thank:

**My friends and family** for all their support.

**Roderick Edwards,** for mentoring, support, encouragement, and patience.

**Sue Whitesides,** for her role as external examiner and her helpful feedback.

**The University of Victoria,** for funding and providing a great learning environment.

# Chapter 1

# Introduction

The mammalian nervous system represents an incredibly intricate and complicated dynamical system. The first step to understanding such a system is to break it down into its simplest parts. In our case, the basic unit of the nervous system is the neuron. The dynamic behaviour of each neuron was originally described as a four-dimensional ordinary differential equation [13]. Since then many alternative models have been developed either to reduce complexity, increase biological realism, or model the behaviour of different variations on the basic neuron [17]. A single nervous system may be comprised of trillions of neurons that interact via biological structures known as synapses. The nervous system is thus seen as a large network of neurons with some connectivity pattern. To make matters worse, a neural network is by nature adaptive. The synaptic efficacy changes in response to the dynamics of the network through the biological processes of long-term potentiation and long-term depression (increasing and lowering efficacy, respectively). Describing a biological neural network in full detail is obviously intractable. However, an alternate approach has yielded some insight as well as countless practical applications. This approach is that of the artificial neural network (ANN). An artificial neural network does not seek to produce a biologically accurate model of a neural network, but instead to extract its fundamental dynamic properties and study the resulting object in an abstract setting.

In order to do this one must examine the dynamic properties of a neuron. The state of a single neuron can roughly be described by two phases, resting and spiking. When a neuron is resting it maintains a constant voltage gradient across its membrane. When the neuron is spiking the voltage gradient is increased and an electrical signal propagates down the axon towards a synapse. Upon receiving this signal chemicals called neurotransmitters are released causing a destabilization in the membrane potential of the neighbouring neu-

ron. If the total signal from neighbouring neurons crosses a certain threshold the behaviour of a neuron transitions from resting to spiking. This is the result of a Hopf bifurcation [17].

In the most basic abstraction, neurons are viewed as simple linear threshold units. At each time step, we determine the state of a neuron by taking a weighted sum of the states of its neighbouring neurons and passing this value to a transfer function. These models were originally studied abstractly as modelling logical statements [20]. Since then countless variations of the basic artificial neural network have been studied [9]. Despite there being no single definition for an artificial neural network they all share the same basic principles, a set of neurons with simple dynamic behaviour that communicate locally via a set of weights. Artificial neural networks have proved to be powerful tools in artificial intelligence and machine learning, providing state of the art results in image and voice recognition, computer vision, and other tasks [12][11][18].

Here we provide a definition which encompasses the most prominent kinds of networks, allowing us to study several important features of artificial neural networks under a common framework. Most importantly we are concerned with the computational and dynamic capabilities of artificial neural networks as a class of computational models. Many results regarding the universal approximation of specific neural networks have been around for decades. We go over some of these results for various models before proving several new universal approximation results for related classes of probabilistic artificial neural networks. We conclude with a set of experiments that highlight the possibility of the use of artificial neural networks to provide insight into the biological process behind perception.

# Chapter 2

# Artificial Neural Networks

## 2.1 Dynamics and Stability

A neural network is by nature a highly parallel system in which relatively simple computational units interact locally via a set of connections. In order to properly define an artificial neural network so that every network discussed here will fit, it is necessary to identify the essential properties shared by all such networks. Each network that will be discussed consists of a finite number of nodes, loosely called neurons, which, at each time step, take on real values determined from the previous state of the network along with the network's parameters. In general a network is probabilistic; and in full generality a network can have long-term time dependence. Thus we see an artificial neural network as representing a *Markov kernel* from the measure space of a finite number of previous configurations to the space of current configurations. To be precise, given that $X = M^k$ for some $k$ with $M \subset \mathbb{R}$, a Markov kernel is a map $\kappa : X^m \times \mathcal{B} \to [0, 1]$ with *source* $(X^m, \mathcal{A})$ for some $m$ and *target* $(X, \mathcal{B})$ where $\mathcal{A}$ and $\mathcal{B}$ are $\sigma$-algebras on $X^m$ and $X$ respectively where for any $B \in \mathcal{B}$ the map $x^m \to \kappa(x^m, B)$ is measurable and for every $x^m \in X^m$, the map $B \to \kappa(x^m, B)$ is a probability measure of $(X, \mathcal{B})$. For our purposes $\mathcal{B}$ will always be taken as either the discrete or Borel $\sigma$-algebra and $\mathcal{A}$ will be the product $\sigma$-algebra of $\mathcal{B}$. Intuitively, $k$ is the number of neurons and $m$ is the number of previous states the network depends on. The final property shared by the networks discussed is one of locality. Information passes between neurons only via a set of weighted connections. In other words if there is no connection between two neurons then the state of one neuron cannot affect the state of the other neuron at the next time step. We thus define a neural network as follows.

**Def:** *A Neural Network is defined as a triple $(X, \theta, \kappa_\theta)$ consisting of the following,*

i *a set of neurons (or nodes) $X = X_1 \times ... \times X_n$ with $X_i \subset \mathbb{R}^{k_i}$, $k := \sum_{i=1}^{n} k_i$, where $X_i$ is equipped with the discrete $\sigma$-algebra if it is finite and the Borel $\sigma$-algebra otherwise and $X$ has the product $\sigma$-algebra*

ii *parameters $\theta = (W^{(0)}, ..., W^{(m)}, b) \in (M_{k,k}^{m+1}(\mathbb{R}), \mathbb{R}^k)$, where $i$ is called the time dependence of $W^{(i)}$, $m$ is the time dependence of the network, and*

iii *a Markov kernel $\kappa_\theta$ with source $(X^m, \mathcal{A})$ and target $(X, \mathcal{B})$, where $\mathcal{B}$ is the $\sigma$-algebra on $X$ and $\mathcal{A}$ is the product $\sigma$-algebra, satisfying the property that if $w_{i,j}^{(l)} = 0$ and $x^{(l)}$ and $x^{(l)'}$ are two vectors that differ only in the $i^{th}$ component, then the marginal distributions over the $j^{th}$ component of $\kappa_\theta(x^{(1)}, ..., x^{(l)}, ..., x^{(m)}, \cdot)$ and $\kappa_\theta(x^{(1)}, ..., x^{(l)'}, ..., x^{(m)}, \cdot)$ are equal.*

The distributions represent the transition probabilities of the network given a sequence of previous configurations $x^{(1)}, ..., x^{(m)}$. We will denote these distributions by $\Omega_{\theta, x^{(1)}, ..., x^{(m)}}$.

The last part of the third condition is the locality condition. It states that no information can pass between nodes if the connection between them is $0$. This is really only one half of the locality condition as weights between two nodes should have no immediate effect on a distinct third node. In a single neural network the weights are constant so to formalize this property we must include a restriction on sets of neural networks (which we call families). This will be defined later.

Each neural network has a family of stochastic processes associated with it called the *discrete neural dynamics*. The discrete neural dynamics of a neural network is a family of stochastic processes with each process, $(P^{(t)})_{t=m}^{\infty}$, defined by $P^{(t)} : X \to [0, 1]$, $P^{(t)}(x^{(t)}|x^{(t-1)}, ..., x^{(t-m)}) = \Omega_{\theta, x^{(t-1)}, ..., x^{(t-m)}}(x^{(t)})$, for $t \geq m$. $P_0(x^{(m-1)}, ..., x^{(0)})$ is some initial distribution on $X^m$. Then

$$P^{(t)}(x^{(t)}) = \sum_{x^{t-1}} \left( \prod_{i=m}^{t} P^{(i)}(x^{(i)}|x^{(i-1)}, ..., x^{(i-m)}) \right) P_0(x^{(m-1)}, ..., x^{(0)})$$

Here $x^{t-1}$ refers to a sequence of vectors in $X$ of length $t-1$. In the case that $X$ is continuous the above sum is replaced with an integral.

Although one can study a single neural network in isolation, more often we are interested in the behaviour of a set of neural networks somehow defined by a *parameter space*, $\Theta$. We define a *family* of neural networks to be a set of neural networks with the same number of neurons and time dependence, so a set of the form $\{(X, \theta, \mathbf{\Omega}_\theta)\}_{\theta\in\Theta}$ with $\Theta \subseteq (M_{k,k}^{m+1}(\mathbb{R}), \mathbb{R}^k)$, satisfying the condition that for all $1 \leq l \leq m$, if $\theta, \theta' \in \Theta$ with $\theta$ and $\theta'$ differing only in $w_{i,j}^{(l)}$ then the marginal distributions of $\Omega_{\theta x^{(1)},\dots,x^{(m)}}$ and $\Omega_{\theta' x^{(1)},\dots,x^{(m)}}$ over $x_k$ are equal for all $k \neq i, j$.

The locality condition on the Markov kernel of a neural network states that information cannot directly pass between unconnected neurons. The locality condition on a family of neural networks states that the state of a neuron cannot directly depend on a weight it's not connected to. According to this definition, two members of a family of neural networks can be vastly different even if they share similar parameters. This is highly undesirable when it comes to learning and thus one usually discusses families of neural networks that vary smoothly with their parameters. The most natural smoothness condition will be discussed in Section 2.4.

When using neural networks in practice, the fundamental object of interest is not a single neural network but a family of neural networks. Often we do not know a priori how many neurons a neural network will need. Or, we may be interested in a group of neural networks sharing a similar definition but without the same number of neurons. For these purposes we discuss a *class* of neural networks as a union of families. Most often, a class will contain neural networks that are defined in a similar manner.

In the case that the parameter space is a manifold we can see that a family of neural networks is really a sort of manifold in which every point is a Markov kernel. This type of manifold is referred to as a neuromanifold and it comes naturally equipped with a Riemann metric in the form of the *Fischer information*. The geometry of the neuromanifold can yield useful information when it comes to learning [2][3][4]. This will also be discussed in Section 2.4.

A *Deterministic Neural Network* is a neural network for which $\mathbf{\Omega}_\theta$ consists entirely of point-mass functions. In other words the Markov kernel defining the network only maps to values of $1$ or $0$. This is sometimes called a *deterministic Markov kernel*. For the case of a deterministic neural network we will generally choose $P_0$ to be a point mass distri-

bution for some initial state $x^{(0)}, ..., x^{(m-1)}$. In this case we see that the neural dynamics reduce to a dynamical system with flow $f^t(x) = x^{(t)}$ where $x^{(t)}$ is the unique vector with $P^{(t)}(x^{(t)}) = 1$ in the neural dynamics with initial distribution $P_0(x^{(m-1)}, ..., x^{(0)}) = 1$. For a deterministic neural network we define the *transfer function* as the function $f : X^m \to X$ with $f(x_1, ..., x_m) = x$ where $x$ is the unique vector with probability one in the distribution $\Omega_{\theta, x_1, ..., x_m}$.

Given a family of neural networks we are often trying to find a particular neural network in the family that can accomplish a task to a certain precision. To facilitate this, a learning rule is used. A learning rule moves through the parameter space $\Theta$ of a family of networks in an attempt to converge to an optimal solution for our problem. A learning rule will use a variety of information, including sets of training vectors, loss functions, and the geometry of the parameter space to select a certain value for the parameters at each time step. A learning rule, $G$, can be formalized as a set of probability density functions (PDFs) on $\Theta$, $\{G_{\theta, x^{(1)}, ..., x^{(n)}}\}_{\theta, x(1), ..., x^{(n)}}$. Given a set of *training vectors* $V \subset X$ and a sequence of training sequences $\{v_{1,i}, ..., v_{n,i}\}_i, v_{j,i} \in V$ we define the *learning dynamics* as a stochastic process $(G^{(t)})_{t=0}^{\infty}$ with $G^{(t)}(\theta^{(t)}|\theta^{(t-1)}) = G_{\theta^{(t-1)}, v_{1,t-1}, ..., v_{n,t-1}}(\theta)$. $G^{(0)}$ is defined as the point mass distribution for some initial network $\theta$. Despite the possibility of using a formal definition, it is usually more appropriate to think of a learning rule informally as an algorithm that takes in a certain amount of data, along with a starting network, and outputs some other network in the family whose behaviour is, one hopes, closer to the desired behaviour. Often a learning rule will compute the neural dynamics for a finite $t$ in order to get more information. Learning that utilizes the neural dynamics in this way is called *online learning*.

The ability of a neural network to solve a given problem then depends on two things. First, it depends on the existence of a network that can solve the problem. Second, it depends on the existence of a learning rule that can find this network given an appropriate initial condition. Note that the phrase "appropriate initial condition" is somewhat ambiguous, as what is deemed appropriate depends on the situation. It is certainly true that given the existence of a neural network, $N$, approximating a solution to our problem we could merely define a trivial learning rule that at each time step assigns a probability of $1$ to the PDF of $N$ and $0$ to every other distribution. Such a procedure would be of little practical use. Ideally we would like the learning rule to converge to an optimal solution for any initial condition although this may not always be feasible.

Often a neural network will lack the computational power to implement some desired behaviour. One possible way of getting around this is to add *hidden neurons*. These are neurons whose sole purpose is to add computational complexity to a neural network. Furthermore, often we have distinct *input neurons* and *output neurons*. In this case we want to partition $X$ into input, output and hidden neurons. The idea is we are only interested in the initial distribution of the input neurons and the following distributions of the output neurons. The hidden neurons are merely there to increase computational power. In the case where we have input and output neurons we examine the stochastic process marginalized over the output neurons.

One of the most important traits a neural network can have is that of *stability*. Stability of a network roughly refers to the convergence of the neural dynamics to a stationary distribution. In general, however, this may depend on the choice of initial distribution $R_0$. One can easily see that, in the case of a deterministic neural network, we have that $R_0$ corresponds to the initial state of the machine and the convergence of the neural dynamics corresponds to convergence to a point $y$. With this in mind we introduce the following definition.

**Def:** *A Neural Network on $X$ is stable on a set of distributions $\mathbf{P}_0$ if there exists a set of distributions $\mathbf{P}$ such that for all $P_0 \in \mathbf{P}_0$, there exists $P \in \mathbf{P}$ such that the neural dynamics with initial distribution $P_0$ satisfy the property $\lim_{t\to\infty} P^{(t)}(x) = P(x)$ for all $x \in X$. A network is globally stable if it is stable on the entire set of distributions on $X$.*

This is a fairly general definition but we will exclusively be interested in two cases. In the case of a probabilistic network, we will be interested in the case that $\mathbf{P}$ consists of a single distribution. In the case of a deterministic network we will want $\mathbf{P}$ to consist of point mass distributions indexed by $X$. In this case, if the network is globally stable, $\mathbf{P}$ defines a mapping $F : X \to X$ by $F(x) = y$ iff given an initial point-mass distribution $P_0$ with $P_0(x) = 1$, the discrete neural dynamics have a limiting point-mass distribution $P_x \in \mathbf{P}$ with $P_x(y) = 1$. We might consider stable deterministic networks in which $F$ is defined only on a subset of $X$ in which case the neural dynamics have a stable basin of attraction. According to our definition this is sufficient for a network to be stable but not globally stable. In both cases the set $\mathbf{P}$ gives a single function $F$ or a single distribution $P$. A network that is not stable is called *unstable*.

To make the above definition more clear we will define some classic neural networks within this formalism.

## 2.2 The Hopfield Network

The Hopfield network is a class of deterministic neural networks with one-step time dependence, $X = M^n$. $M = \{-1, 1\}$, $f_i(x) = g(\sum_j w_{i,j}^{(1)} x_j + b_i)$. Here $f_i$ denotes the $i^{th}$ component of the transfer function and $w_{i,j}^{(1)}$ is the $i, j^{th}$ component of $W^{(1)}$. The function $g : \mathbb{R} \to M$ is the step function defined by $g(x) = 1$ if $x \geq 0$ and $g(x) = -1$ if $x < 0$. One may consider a continuous-state variation of the Hopfield network by replacing $g$ with any bounded, continuous, non-linear, increasing function. The parameter space of each family of Hopfield networks is required to satisfy the conditions $\forall i, j : w_{i,j}^{(1)} = w_{j,i}^{(1)}, w_{i,i}^{(1)} = 0$. In the original formulation of the Hopfield network we take $g$ to be the unit step function with threshold $0$ and $M = \{-1, 1\}$. Often the Hopfield network is updated asynchronously, i.e. one unit at a time is chosen to update its value. This can be defined by choosing a probability distribution, $\omega$, over the number of nodes. The transition probabilities of the network are then defined as $x_i \leftarrow f_i(x)$ with probability $\omega(i)$ and $x_i \leftarrow x_i$ with probability $1 - \omega(i)$.

The Hopfield network was originally formulated as a model of an associative memory [14]. An associative memory is one in which memory is accessed based on its content rather than an address, the idea being that given a fragment of a particular input the associative memory will retrieve a specific desired memory. In this context it becomes clear that an associative memory can be seen as a map $F : \mathbb{R}^n \to \mathbb{R}^n$ with a countable range and ideally the property that $F(x) = y \iff x \in A_y$ for some $A_y \subset \mathbb{R}^n$. One can see how the Hopfield network might be constructed in a way to give an associative memory by choosing appropriate parameters so the memory $y$ has a basin of attraction consisting of similar patterns. In order to show that the asynchronous (where only a single unit is updated at a time) Hopfield network forms an associative memory we must first ensure that it is stable.

**Theorem: 2.1** *The asynchronous Hopfield network is stable.*

*Proof:* The proof is given by showing that the energy function $E(x) = -\frac{1}{2} \sum_{i,j} w_{i,j}^{(1)} x_i x_j +$

$\sum_i b_i x_i$ is a Lyapunov function. That is, we must show that the above function is bounded below and that $E(f(x)) \leq E(x)$ for all $x \in X$. That $E$ is bounded comes immediately from the fact that $M$ is finite. Now consider $E(f(x)) = -\frac{1}{2}\sum_{i,j} w_{i,j}^{(1)} f_i(x) f_j(x) + \sum_i b_i f_i(x)$. If $i$ is the unit that is updated, this can be written

$$E(f(x)) = -\left( (b_i + \sum_j w_{i,j}^{(1)} x_j) f_i(x) + \frac{1}{2} \sum_{l,j:l,j\neq i} w_{l,j}^{(1)} x_l x_j + \sum_j b_j x_j \right).$$

But $f_i(x) = -1$ if $b_i + \sum_j w_{i,j}^{(1)} x_j < 0$ and $f_i(x) = 1$ if $b_i + \sum_j w_{i,j}^{(1)} x_j \geq 0$ so in either case $-(b_i + \sum_j w_{i,j}^{(1)} x_j) f_i(x)$ remains the same or decreases. Assuming that for all $i$, $\omega(i) \neq 0$, given enough time this function will reach a local minimum.■

Learning in the Hopfield network is Hebbian, meaning that neurons whose values correlate have the connection between them strengthened and those whose values do not correlate have the connection between them weakened [9]. Given a number of patterns $x^{(1)}, ..., x^{(n)}$ that we want to be the fixed points of our associative memory, we can define a deterministic learning rule as follows: $\Delta w_{i,j}^{(1)} = x_i^{(i)} x_j^{(i)}$. Starting with $w_{i,j}^{(1)} = 0$ after $n$ steps we end up with $w_{i,j}^{(1)} = \sum_{i=1}^{n} x_i^{(i)} x_j^{(i)}$. The learning rule works by making the vectors $x^{(1)}, ..., x^{(n)}$ minima of the energy function. Unfortunately there are several drawbacks with this procedure. First is the existence of spurious minima, that is local minima other than one of $x^{(1)}, ..., x^{(n)}$. The second is one of memory capacity, that is the number of patterns it can store. A Hopfield network with a fixed number of neurons only has a limited number of vectors that can exist as stable states in the network [1][9]. For these reasons, the basic Hopfield network is rarely used in practice today; however, countless variations of the Hopfield network have been developed to address the deficiencies of the basic model.

## 2.3 Feed-Forward Networks

One key aspect of the Hopfield network is that it is *recurrent*. A recurrent network is a network that has a cycle in at least one of its weight graphs. In contrast to recurrent networks, *feed-forward* networks contain no cycles. A feed-forward network can thus be divided into layers in which all connections go from one layer to a higher layer. More precisely, a feed-forward network is a network with neurons $X = X_1, ..., X_n$ with the weight matrix, $W^{(l)}$,

for each time dependence of the network, $l$, satisfying the following two properties: (1) for $x_i \in X_k$ with $k > 1$, $w_{i,j}^{(l)} = 0$ if $x_j \in X_m$ with $m \leq k$, and for $x_i \in X_1$, $w_{i,i}^{(1)} \neq 0$, and $w_{i,j} = 0$ if $x_i, x_j \in X_1$ with $x_i \neq x_j$, and (2) we require the transition distributions of a feed-forward network to satisfy $\Omega_{\theta,x^{(1)},\ldots,x^{(m)}}(x_i^{(t)} = x_i^{(1)}) = 1$ for $x_i \in X_1$. Property 1 states that the state of the nodes in every layer other than the first depends only on the state of the nodes in previous layers. Property 2 requires that the nodes in the first layer remain in the initial state for all time. If there were no connections between nodes in the first layer they would be unable to communicate and it would be impossible to ensure the condition on the transition distributions. This is why we require $w_{i,i} = 1$ for $x_i \in X_1$.

Observe that every feed-forward network is stable. This can be seen by considering the locality condition on the distributions of a neural network. The first layer is constant for all $t \geq 1$. Then if $m$ is the time dependence of the network, the distribution of the second layer will be stationary for $t \geq m$ since it only depends on the conditional distribution given the state of the first layer, which is constant. The distribution of subsequent layers is then given as a product of conditional distributions. Since there are a finite number of layers, eventually after enough time this distribution will be stationary.

In some cases we may wish the input layer to be given a stationary distribution rather than a constant input. The same definition can be used for this case if the condition $w_{i,i}^{(l)} = 0$ for $x_i \in X_1$ is removed. In the deterministic case, we can immediately see a feed-forward network as representing a function, with the first layer being the input layer and the final layer being the output layer.

The canonical example of a feed-forward network is the *perceptron*. The perceptron is a deterministic feed-forward network with $n$ layers and 1-step time dependence. The first layer is the set of input neurons, the last layer is the set of output neurons, and the remaining layers are the hidden neurons. There are many variants of the perceptron using various transfer functions and state spaces. A common example of an $n$-layer perceptron is given by $X = X_1 \times \ldots \times X_n$, $X_i = [-1, 1]^{k_i}$, where the weight matrices satisfy the feed-forward condition with $w_{i,i} = 1$ for $x_i \in X_1$, the transfer function for nodes $x_j$ in a hidden layer $l$ is given by $f_j(x) = \tanh(b_j + \sum_{i:x_i \in X_{l-1}} w_{i,j} x_i)$, and the transfer function for nodes $x_j$ in the first and final layer is linear. Since $w_{i,i} = 1$, by setting the biases to 0 for each node in the first layer, the value of the first layer remains constant for all time giving us an obvious

input/output mapping from the first to final layer. The original use of the perceptron was for binary classification. That is, given a function mapping $X_1$ to $\{0, 1\}$, we wish to find a perceptron implementing this function. Originally a 2-layer perceptron was used, although this turns out to be insufficient to implement an arbitrary binary classifier, hence the use of non-linear hidden units [21].

Given a function $g : I \to O$ we wish to find a perceptron with input nodes $I$ and output nodes $O$ that approximates $g$. To do this we use a learning rule called *backpropagation*. Given a loss function, backpropagation calculates the gradient of the loss function with respect to the parameters of the network. The basic loss function used for perceptrons is defined as $l : X \times Y \times \Theta \to [0, \infty)$, $l(x, y, \theta) = \frac{1}{2}\sum_i (y_i - y_i')^2$ where $y_i'$ is the value of the $i'$th output node of the perceptron with input $x$ and parameters $\theta$, and $y_i$ is the $i^{th}$ component of $g(x)$. Then by minimizing $L(\theta) = \sum_{(x,g(x))} l(x, g(x), \theta)$ the function given by the perceptron becomes close to the desired function pointwise. The learning rule is then defined by $\Delta w_{i,j} = -\alpha \frac{\partial L(\theta)}{\partial w_{i,j}}$ where $\alpha$ is a positive constant called the learning rate. The $\Delta b_i$ are defined in the same way. In backpropagation we begin by running a forward pass on an input and then using the values of the network to calculate the gradient of the loss function with respect to the weights going to the output layer. From these values we pass back to the second to last layer and compute the gradient of the loss function with respect to the weights going into this layer. We continue on in this fashion until we get to the first layer. A similar procedure works for the biases. The full details of the algorithm including the calculations involved can be found in [26]. Backpropagation can be summarized as follows

---

1: **procedure** BACKPROPAGATION
2: Begin with training vector $x$
3: Compute the state of the output nodes, $y$, using $x$
4: Compute the error $\frac{1}{2}(g(x) - y)^2$
5: Use this to compute $\Delta w_{i,j}$ for each $w_{i,j}$ one layer at a time moving backwards starting at the second to last layer.

---

The fact that the perceptron is stable makes modeling data with time dependence a little unnatural. Perceptrons are best suited to classification problems whereas more often when dealing with time series we are interested in being able to predict the next vector in a sequence. Despite this, backpropagation is a useful way of calculating the gradient of a loss function and we would like to be able to use a similar algorithm in the unstable

context. Backpropagation has been adapted to a dynamic setting independently by various researchers in various contexts. See [26] for example. The modified algorithm is called *backpropagation through time*. Backpropagation through time "unfolds" a recurrent network by making $t$ copies of the network where $t$ is the number of time steps in the sequences we wish to model. The connections of the recurrent network become feed forward where each layer represents the state of the network at time $t$. Given a desired sequence of vectors possibly along with an input sequence, to do a forward pass we simply run the network starting at the first layer with the first input vector and then move to the second layer whose input is given by the neurons in the first layer along with, possibly, the second input vector. Continuing on this way we end up with a sequence of neural states which we can compare to our desired output. Then starting at the last time step we perform backpropagation on the whole network using the desired output vector at the last time step before moving on to the second last time step and performing gradient descent on the last $t-1$ layers of the network using the desired output at time $t-1$. This gives an efficient way to train unstable networks over finite time sequences. Given that backpropagation can be adopted in a wide variety of cases, we hope to be able to develop some general learning algorithms for neural networks. This is done by looking at the geometric structure of the parameter space of a class along with a loss (or energy) function to be optimized.

## 2.4 Kullbeck-Leibler Divergence and the Neuromanifold

Suppose that for a globally stable network on $X$ with a single limiting distribution, $P$, there is some distribution $R$ on $X$ that we wish the limiting distribution of our network to approximate. We would like to have a way of measuring how well $P$ approximates $R$. This situation arises commonly in information theory which gives us the appropriate way of measuring how good our approximation is. This is the Kullbeck-Leibler (KL) divergence defined as follows

$$KL(R||P) = \sum_x R(x) \log \left( \frac{R(x)}{P(x)} \right).$$

Note that the KL-divergence is not a metric: it is not symmetric and it does not satisfy the triangle inequality. Intuitively, the KL-divergence measures the information lost when $P$ is used to approximate $R$. The KL-divergence is only defined if $R(x) = 0$ whenever $P(x) = 0$. In this context we define $0 \log 0 = 0$. For the case that $X$ is continuous, the above sum is replaced with an integral. In order to minimize this, we would like to use gradient descent. This can be done assuming that, for a family of networks, the function

$\theta \to P(x)$, where $P$ is the limiting ($t \to \infty$) distribution of the network given by $\theta$, is differentiable. We can extend this to the more general case where we have a finite number of limiting distributions, $P_i$, along with a set of corresponding desired distributions, $R_i$. Then summing up the KL-divergences, $\sum_{i=1}^{k} KL(R_i||P_i)$, gives us a suitable quantity to attempt to minimize. Performing gradient descent on the KL-divergence gives us a very general learning rule which should in theory work on any globally stable network with a finite number of limiting distributions that satisfy the above smoothness condition and with $P_i(x) = 0$ only if $R_i(x) = 0$. The last condition is to ensure that the KL-divergence is defined. It is important to note that there is a large class of networks that will not satisfy this condition, namely deterministic neural networks.

There are other reasons that this approach will not work in practice. For example, we do not always have a known distribution $R(x)$ that we want to approximate. More often we have a series of training vectors and (possibly) a corresponding series of desired outputs. A more general approach is to define a loss function as in the multilayer perceptron. We define a loss function as a differentiable function $l$, $l : X \times \Theta \to [0, \infty)$, which takes as input a training vector $x$, along with the parameters of a network $\theta$, computes $P(x)$ and compares it to a desired output $y$ in some way. Then, defining $L(\theta) = \sum_{(x,y)} l(x, y, \theta)$, the objective becomes to minimize this function, which can again be done with gradient descent. The learning is called *supervised* if we are provided with the desired output $y$ and *unsupervised* if we are not.

We have already seen loss functions for deterministic networks in the form of the energy function of a Hopfield network and the error of the perceptron. Although the energy function of the Hopfield network can take negative values, it is bounded below and hence behaves the same way as a loss function. The perceptron was an example of supervised learning whereas the Hopfield network was an example of unsupervised learning. For a probabilistic network, if we are given a distribution $R$ we wish to approximate, then we may use supervised learning by minimizing the $KL$-divergence; otherwise we define an alternate loss function. The most common loss function is the log likelihood, $l(x, \theta) = \log P(x)$ where $P$ is the limiting distribution of the network with parameters $\theta$.

Gradient descent works when the parameter space of a network is Euclidean (in our case just $\mathbb{R}^n$). However, in many examples, including the previous two, the parameter space is

actually a manifold embedded in Euclidean space. In this case the smoothness condition on $\theta \to P(x)$ is in fact a smoothness condition $\xi \to P(x)$ where $\xi \in \Xi$ and $\Xi$ is a local coordinate system for $\Theta$. This is the smoothness condition alluded to earlier. For globally stable networks with a single limiting distribution the neuromanifold can be seen as a *statistical manifold*, in other words a set of distributions $\{P;\theta\}_\Theta$ parameterized by $\theta$ where $P$ is the limiting distribution of the network given by $\theta$. A manifold, $M$, is called a ***Riemann manifold*** if it is a smooth manifold equipped with a set of inner products, $g(p)$, $p \in M$ defined on the tangent space of $M$ at $p$. Here $g(p)$ is required to vary smoothly in $p$, in other words if $X, Y$ are vector fields on $M$ then the map $p \to g(p)(X(p), Y(p))$ is smooth. The metric can be defined as a matrix, $G(p)$, with $g(p)(X, Y) = X^\top G_p Y$. A statistical manifold can be given a Riemman metric called the Fisher information defined by

$$G_{i,j}(\theta) = \sum_x \frac{\partial \log P(x;\theta)}{\partial \theta_i} \frac{\partial \log P(x;\theta)}{\partial \theta_j} P(x;\theta).$$

Again the sum is replaced by an integral in the continuous case. In this case gradient descent does not actually give the steepest direction. Instead one must make use of the Fisher information [2]. A technique called *natural gradient descent* modifies the standard algorithm in order to take into account the Fisher information. The natural gradient is calculated as follows:

$$-\tilde{\bigtriangledown} L(\theta) = -G^{-1}(\theta) \bigtriangledown L(\theta),$$

where $-\tilde{\bigtriangledown} L(\theta)$ is the direction of steepest descent of $L$ with respect to the parameters $\theta$. $G^{-1}(\theta)$ is the inverse of the Fisher information matrix and $\bigtriangledown L(\theta)$ is the conventional gradient.

Unstable networks have no limiting distribution on which to define the Fisher information. Instead they have a set of conditional distributions forming a product of statistical manifolds. This structure was examined by [19]. An alternative approach is to consider the distribution of an unstable network over a finite number of time steps for a given initial distribution. This then becomes equivalent to the stable case and will be our preferred approach when we train unstable networks later on.

## 2.5 The Boltzmann Machine

So far the examples we have looked at have been deterministic. Now we will examine the Boltzmann machine, which can essentially be seen as the probabilistic analog to the Hopfield network. We define the state space of the Boltzmann machine to be $X = \{0,1\}^n$ with the parameter space satisfying the same conditions as the Hopfield network. The transition probabilities for the Boltzmann machine are defined as follows, $\Omega_{(W^{(1)},b),x'}(x_i = 1) = \sigma(\sum_j w^{(1)}_{i,j} x'_j + b_i)$, where $x_i$ is the $i^{th}$ component of $X$ and $\sigma : \mathbb{R} \to [0,1]$ is the logistic function $\sigma(z) = \frac{1}{1+e^{-z}}$. The Boltzmann machine is partitioned into hidden and visible units. This is an input/hidden/output partition where the input and output units are the same and labeled visible units. The set of hidden and visible units are denoted $H$ and $V$ respectively. Like the Hopfield network, the Boltzmann machine admits the energy function $E(x) = -(\frac{1}{2}\sum_{i,j} x_i w^{(1)}_{i,j} x_j + \sum_i b_i x_i)$. From this energy function we can define a *Boltzmann distribution* by $P(x) = e^{-E(x)}/Z$ where $Z$ is the normalization factor defined by $Z = \sum_x e^{-E(x)}$. A Boltzmann distribution is a type of probability distribution arising from statistical mechanics describing the probability that a system is in a certain state given a function describing the energy of each state of the system. In our context we wish to show that our Boltzmann distribution is the limiting distribution of the Boltzmann machine. In other words the Boltzmann machine is stable with $\mathbf{P} = \{P\}$ where $P$ is the Boltzmann distribution.

**Theorem 2.2:** *The Boltzmann machine is globally stable with its Boltzmann distribution as its limiting distribution*

*Proof:* First we will show that the Boltzmann distribution is a stationary distribution for the Markov chain defined by the neural dynamics of the Boltzmann machine and then we will prove this Markov chain is *ergodic* and hence must converge to its stationary distribution. To see that the Boltzmann distribution is stationary it is enough to see that the Boltzmann distribution satisfies *detailed balance*. That is, given a Markov chain with transition probabilities $M_{x,x'}$ (this being the transition probability from $x$ to $x'$) and a distribution $P$, we have that

$$M_{x',x}P(x') = M_{x,x'}P(x).$$

Or equivalently

$$\frac{M_{x',x}}{M_{x,x'}} = \frac{P(x)}{P(x')}.$$

To see that this implies that $P$ is stationary, simply sum over all states $x$. We get

$$\sum_{x'} M_{x',x} P(x') = \sum_{x'} M_{x,x'} P(x) = P(x) \sum_{x'} M_{x,x'} = P(x)$$

which is the definition of a stationary distribution. In our case we have that $\frac{P(x)}{P(x')} = \exp(E(x') - E(x))$. Note that since each node is sampled independently of the others and since $\exp(x - x'') = \exp(x - x')\exp(x' - x'')$ it suffices to prove this for the case that $x$ and $x'$ differ in only a single node. Suppose we have that $x_i = 1$ and $x'_i = 0$, then $M_{x',x} = \sigma(\sum_j w^{(1)}_{i,j} x'_j + b_i) = \sigma(E(x') - E(x))$. Then using the fact that $1 - \sigma(z) = \sigma(-z)$ we calculate that

$$\frac{M_{x',x}}{M_{x,x'}} = \frac{1 + \exp(E(x') - E(x)}{1 + \exp(E(x) - E(x'))}$$

$$= \exp(E(x') - E(x)) \frac{\frac{1}{\exp(E(x') - E(x))} + 1}{1 + \exp(E(x) - E(x'))}$$

$$= \exp(E(x') - E(x))$$

so the Boltzmann distribution satisfies detailed balance and is hence stationary under the network.

A finite state Markov chain is ergodic if it is irreducible and has an aperiodic state. A Boltzmann machine with $n$ nodes has $2^n$ states so it is a finite state Markov chain. The fact that it is irreducible and aperiodic comes from the fact that $\sigma(\sum_j w^{(1)}_{i,j} x'_j + b_i) > 0$. This implies that the transition probabilities from any state of the Boltzmann machine to any other state are non-zero and hence the machine is irreducible and aperiodic. For any initial distribution, the Boltzmann machine will converge to a unique stationary distribution which as we have seen must be the Boltzmann distribution. ■

The Boltzmann machine is most often used to model a probability distribution with its Boltzmann distribution. Given a specific distribution on the visible units $R$ that we wish to model, the problem becomes how to find weights so that the conditional distribution, $P_v$, over the visible units of $P$ is close to $R$. Following the discussion in the previous section, we wish to minimize $KL(R||P_V)$. To do this we calculate the gradient with respect to the

parameters. Computing this explicitly we get

$$\frac{\partial KL(R||P_V)}{\partial w_{i,j}^{(1)}} = (E[x_i x_j]_{clamped} - E[x_i x_j]_{free}).$$

The derivation of this formula can be found in Hertz et al. (1991). $E[x_i x_j]_{clamped}$ is the expected value of the product of nodes $x_i x_j$ where the visible units are drawn from the distribution $R$ and $E[x_i x_j]_{free}$ is the expected value of $x_i x_j$ in the Boltzmann distribution. With this computation we can use gradient descent to minimize the KL-divergence. Given parameters $\theta'$, at each step the learning rule sets $\theta = \theta' - \alpha \frac{\partial KL(R||P_V)}{\partial \theta'}$. In other words we define a deterministic learning rule on a family of Boltzmann machines by $G_{\theta'}(\theta) = 1$ if $\theta = \theta' - \alpha \frac{\partial KL(R||P_V)}{\partial \theta'}$ and 0 otherwise. Here $\alpha$ is some small real number called the *learning rate*. This learning rule turns out to be computationally difficult in practice since in order to compute $P(x)$ one must sum over the total number of states of the machine, which is exponential in the number of nodes. Note that the expectation of the clamped distribution requires one to compute $P(h|v)$ instead of $P(v, h)$ (since the clamped distribution by definition is $R(v)P(h|v)$ whereas the free distribution is simply $P(v, h)$). This is still difficult in a general Boltzmann machine as one still must sum over every hidden state. However, the computations of $P(h|v)$ and $P(v|h)$ can be simplified greatly if the hidden units and visible units are conditionally independent. That is, each of the hidden nodes are mutually independent and each of the visible nodes are mutually independent. This condition can be imposed by adding an additional restriction to the parameter space. This more restricted class of neural networks is called the *restricted Boltzmann machine* (RBM). A restricted Boltzmann machine is simply a Boltzmann machine with the additional condition, $w_{i,j}^{(1)} = 0$ if $x_i, x_j \in H$ or $x_i, x_j \in V$ (see Fig 2.1). This allows $E[x_i x_j]_{clamped}$ to be computed much more efficiently as the distributions on the hidden units are now independent. However, $E[x_i x_j]_{free}$ is still difficult to compute.

In general we do not have a specific distribution in mind but instead have a set of training vectors we want the machine to assign a high probability. This is done by assigning as a loss function the negative log-likelihood of a training vector under the model. Given a training vector $v$ we wish to calculate the gradient of the negative log probability of observing $v$ in our Boltzmann machine. To do this we first define $F(v) = -\log \sum_h e^{-E(v,h)}$. Then we have $P(v) = \frac{e^{-F(v)}}{Z}$ and $-\log P(v) = F(v) - \log(Z)$. Calculating the gradient

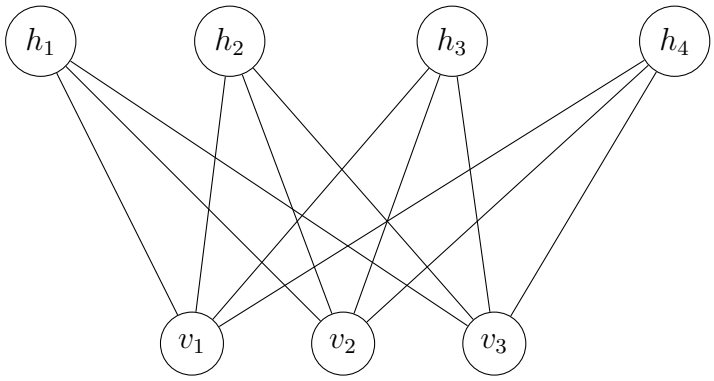


Figure 2.1: *An example of a restricted Boltzmann machine with* $3$ *visible units and* $4$ *hidden units.*

gives us

$$-\frac{\partial \log P(v)}{\partial w_{i,j}^{(1)}} = \frac{\partial F(v)}{\partial w_{i,j}^{(1)}} - \frac{\partial}{\partial w_{i,j}^{(1)}} \log \left( \sum_{v,h} \exp(-E(v,h)) \right)$$

$$= \frac{\partial F(v)}{\partial w_{i,j}^{(1)}} - \frac{1}{Z} \left( \sum_{v,h} \exp(-E(v,h)) \left( -\frac{\partial E(v,h)}{\partial w_{i,j}^{(1)}} \right) \right)$$

$$= \frac{\partial F(v)}{\partial w_{i,j}^{(1)}} - \sum_{\overline{v}} P(\overline{v}) \frac{\partial F(\overline{v})}{\partial w_{i,j}^{(1)}}.$$

To calculate the derivative of $F(v)$ we note that

$$F(v) = -\log \left( \exp(c^\top v) \sum_h \exp(\sum_{i,j} w_{i,j}^{(1)} v_i h_j + \sum_j b_j h_j) \right)$$

$$= -c^\top v - \log \left( \sum_h \exp \left( \sum_j \left( \sum_i w_{i,j}^{(1)} v_i h_j + b_j h_j \right) \right) \right)$$

$$= -c^\top v - \log \left( \sum_h \prod_j \exp \left( \sum_i w_{i,j}^{(1)} v_i h_j + b_j h_j \right) \right).$$

We can write $\sum_h$ as $\sum_{l=0}^{1} \sum_{h:h_j=l}$ in which case we can factor out the term depending on $j$. Repeating this for each hidden node gives us

$$= -c^\top v - \log \left( \prod_j \left( 1 + \exp(\sum_i w_{i,j} h_j v_i + b_j) \right) \right)$$

$$= -c^\top v - \sum_j \log(1 + \exp(b_j + W_j^{(1)} v)).$$

Here $c$ is the vector of visible biases, $b$ is the vector of hidden biases, and $W_j^{(1)} v$ is shorthand for $\sum_i w_{i,j} h_j v_i$.

$$\frac{\partial F(v)}{\partial w_{i,j}^{(1)}} = \frac{\exp(b_i + W_i^{(1)} v) v_j}{1 + \exp(b_i + W_i^{(1)} v)} = \sigma(b_i + W_i^{(1)} v) v_j = P(h_i = 1 | v_j) v_j.$$

Given a set of training vectors $V = \{v_1, ..., v_n\}$ we form a sequence $\{v_i\}_{i=1}^{\infty}$ with $v_i \in V$. Then we define the learning rule as $G_{\theta', v}(\theta) = 1$ if $\theta = \theta' + \alpha \frac{\partial \log p(v)}{\partial \theta'}$ and $G_{\theta', v}(\theta) = 0$ otherwise. Again $\alpha$ is the learning rate. The first term in the gradient is called the positive phase and the second the negative phase. This terminology comes from the fact that the positive phase raises the probability of observing $v$ whereas the negative phase decreases the probability of samples from the Boltzmann machine. In a general Boltzmann machine computation of the positive phase is difficult as it requires one to compute $P(h|v)$, which is intractable. However, in the restricted Boltzmann machine the hidden units are mutually independent and hence $P(h|v)$ becomes easily computable, which allows for the positive phase to be computed in a straightforward manner. Unfortunately the negative phase is still difficult to compute in practice as it requires computation of $P(v)$. To get around this, we will approximate the gradient by sampling from the Boltzmann distribution.

Although calculating $P(v, h)$ explicitly is difficult, one can sample from the Boltzmann distribution using *Markov chain Monte Carlo*, that is starting the Boltzmann machine in an initial state, we run the machine step by step until it converges to its Boltzmann distribution (as guaranteed by Theorem 2.3). In a restricted Boltzmann machine the set of visible nodes is conditionally independent of the hidden nodes. This makes it easy to implement *Gibbs sampling*. The full procedure is outlined as follows. Starting with a vector $v$ from the training data, we sample from $P(h^{(t)} | v^{(t-1)} = v)$ which is easily done as $P(h_i^{(t)} = 1 | v) = \sigma(\sum_j w_{i,j}^{(1)} v_j + b_i)$ and all hidden units are mutually independent. We then sample from $P(v^{(t+1)} | h^{(t)})$ and continue for $k$ steps until we have a vector $\overline{v}$. Running this procedure for $k \to \infty$ we end up with a sample from our Boltzmann distribution. However, in practice we run for only a finite number of steps. Using this technique we can approximate the gradient of the log-likelihood by sampling from the Boltzmann distribution rather than explicitly computing $P(v)$. This method of learning is called *Contrastive Divergence*

[10]. Rather than define the learning rule as an explicit probability distribution on the parameter space, it is simpler to describe contrastive divergence as an algorithm.

---

1: **procedure** CONTRASTIVE DIVERGENCE
2: Begin with training vector $v^1$
3: Sample $h^1 \leftarrow P_H(\cdot|v^1)$
4: Sample $v^2 \leftarrow P_V(\cdot|h^1)$
5: Repeat $k$ times
6: Set $w_{i,j}^{(1)}$ to $h_i^1 v_j^1 - h_i^k v_j^k$

---

Although contrastive divergence does not follow the gradient of $KL(R||P_V)$, it does follow the gradient of $KL(R||P_V) - KL(P_V||P_{n,V})$ where $P_{n,V}$ is the distribution of the Boltzmann machine after running it for $n$ steps [10]. Contrastive divergence has proven to be effective in practice and will be used later to train Boltzmann machines. Note the above learning rules can be adapted to model the likelihood of several training vectors at a time. Rather than calculating the gradient of the likelihood of a single training vector, we instead define the likelihood of a set of training vectors, $\mathbf{V}$, and weights $\theta$, as $\mathcal{L}(\theta|\mathbf{V}) = \prod_{v \in \mathbf{V}} P(v)$. Taking the logarithm of this yields the marginalized loss function $L = \sum_v l(v, \theta)$ as discussed in the previous section. Here $l(v, \theta)$ is the negative log-likelihood of $v$ under the Boltzmann machine given by parameters $\theta$. Although we only showed the calculations for the derivatives with respect to the weights explicitly, learning rules for the biases are developed in the same fashion.

Neural networks are closely related to probabilistic models called *directed graphical models*. Directed graphical models are diagrams that show the conditional dependence of random variables. Given a joint probability distribution over a set of nodes $X$, a directed graphical model is a graph consisting of a single node for every variable in $X$ and a directed edge from $x$ to $y$ if and only if $y$ depends on $x$. A *deep belief network* is a directed graphical model in which the nodes are divided into layers and every pair of nodes in a single layer is independent. In the undirected case, one can view a deep belief network as a stack of RBMs where each pair of layers of the network consists of a single RBM. With this view, one can train deep belief networks greedily by training an RBM and then using the hidden units of the RBM as the visible units of the next RBM and training a second one "on top of" the existing machine[12].

Now that we have seen several examples of artificial neural networks and their learning

rules we turn to the question of representational power of a class of neural networks. That is, we ask what is the set of functions or distributions that correspond to a network from our class.

# Chapter 3

# Universal Approximation

## 3.1 Universal Approximation for Globally Stable Networks

For globally stable deterministic networks, we can identify each network with a function $F : X \to X$ by associating each point $x \in X$ with the point-mass distribution on $X$ and mapping it to the associated limiting point-mass distribution of the network defined by a single point $y$. $F$ is the mapping $F(x) = y$. The question now is whether or not the set of functions given by a class of globally stable networks is *dense* in the entire set of distributions/functions on $X$. This gives us our first notion of universal approximation. When we say a set of functions $\mathbf{F}$ on $X$ is dense we mean that for any $G : X \to X$, $\exists\{F_i\}_{i=1}^{\infty} \subset \mathbf{F}$ such that for all $x$, $\lim_{i\to\infty} F_i(x) = G(x)$. A class of globally stable deterministic neural networks is then a *universal approximator* if the set of associated functions is dense. Often we may talk of universal approximation for smaller sets of functions. For example, rather than being able to approximate every function on $X$, a class of neural networks may have functions that are dense in the set of continuous or bounded functions on $X$. Note that if the functions are bounded, the definition is equivalent to saying that for all $G : X \to X$ and all $\epsilon > 0$ there exists a function $F$ associated to a neural network in our class such that $||G - F|| < \epsilon$ where $||\cdot||$ is the uniform norm. However, we do not prefer this definition as we do not require in general that $G$ and $F$ are bounded. Often a family of neural networks on $X$ is not a universal approximator but we can add hidden units to increase computational power. In the case that we have an input/output/hidden partition we are looking to approximate functions from the input set to the output set. Here the neurons of the family of neural networks with state space $X$, will be partitioned into input, hidden, and output neurons with state space $I$, $H$, and $O$, respectively. In this case we look at a family of glob-

ally stable neural networks on $X$ whose functions satisfy $\pi_o F(x) = \pi_o F(y)$ if $\pi_i x = \pi_i y$ where $\pi_o$ is projection onto the output nodes and $\pi_i$ is projection onto the input nodes. In this case, given the associated function to a network, $F$, we have a well defined function $\overline{F} : I \to O$ and can use the same definitions to describe universal approximation of functions $G : I \to O$.

For probabilistic networks stable on a set of distributions $\mathbf{R}$, we are interested in the case where the set of limiting distributions consists of a single distribution, $P$. Our notion of density for a set of distributions comes from the Kullback-Leibler divergence. We say that a set of distributions $\mathbf{P}$ is dense in a set of distributions, $\mathbf{R}$, if for each $R \in \mathbf{R}$ and all $\epsilon > 0$, there exists a $P \in \mathbf{P}$ such that $KL(R||P) < \epsilon$. We then say that a family of stable probabilistic neural networks on $X$ is a universal approximator if the set of associated distributions is dense in the set of distributions on $X$. Similar to deterministic networks, we will often add hidden nodes to increase computational power. In this case we look to approximate distributions over the output nodes by the marginal distributions over the output nodes.

An important question when dealing with universal approximation is the number of hidden nodes required for a class of networks to be a universal approximator. Ideally we would like a universally approximating class of networks to have some finite (preferably small) upper bound for the number of hidden nodes. A family of neural networks with an input/output/hidden partition with $k$ hidden nodes will be called a $k$-family. Given a particular state space $X$ and a particular $k$, the question then becomes, is there a $k$-family that is a universal approximator for functions/distributions on $X$. It is easy to construct examples of types of neural networks with no $k$-family that is a universal approximator. For example the single-layer perceptron is unable to implement the XOR function and is therefore not a universal approximator. Thus the class of perceptrons contains no $0$-family that is a universal approximator.

It turns out that we have already seen examples of universal approximators. For the deterministic case, although the perceptron with no hidden units is not a universal approximator, the perceptron with a single hidden layer is a universal approximator for continuous functions on a compact set [15]. Probabilistically, both the Boltzmann machine and the Restricted Boltzmann machine are universal approximators. In fact the RBM has a $k$-family that is a universal approximator for distributions on $X$ where $X = \{0, 1\}^n$. This is stated

precisely as follows.

**Theorem 3.1:** *The $2^n$-family of Restricted Boltzmann machines is a universal approximator for distributions on $\{0,1\}^n$.*

*Proof:* We will provide a sketch of the proof here. The full proof can be found in [24]. We begin the proof by showing that we can add a hidden node that will increase the probability of a certain vector in the Boltzmann distribution while uniformly decreasing the probability of every other vector by some multiplicative factor. We start with an RBM whose weights and biases are set to $0$ and a target distribution, $R$, we order the vectors with non-zero probability $v_1, ..., v_m$ in $R$ so that $R(v_1) \leq ... \leq R(v_m)$. For any RBM with distribution $P$, it can be shown that for each vector $v_i$ with non-zero probability, we can add a hidden node with parameters so that $\tilde{P}(v_i) = \frac{(1+\exp(\lambda))P(v_i)}{1+exp(\lambda)P(v_i)}$ and for $v_k \neq v_i$ $\tilde{P} = \frac{P(v_k)}{1+\exp(\lambda)P(v_i)}$ where $\lambda$ is arbitrary and $\tilde{P}$ is the distribution resulting from adding the hidden node. With this in mind, we begin with the Boltzmann machine with no hidden units, which gives a uniform distribution over the visible units. Then starting at $v_1$ we add a hidden node with parameters so that $P(v_1)$ is close to $1$ and the remaining vectors $v_2, ..., v_m$ have uniform probability. This can be done in the above way so that the probability of each $v_i$ is determined entirely by some $\lambda_1$. We then add an additional node in the same manner determined by some $\lambda_2$. Since we can use this procedure to increase $P(v_2)$ arbitrarily close to $1$ (and smoothly), we can choose $\lambda_2$ in such a way that $\frac{P(v_2)}{P(v_1)} = \frac{R(v_2)}{R(v_1)}$. We continue to add hidden nodes at each time step with parameters defined so that $\frac{P(v_k)}{P(v_{k-1})} = \frac{R(v_k)}{R(v_{k-1})}$. Note that at each step, $j$, if $i < j$ then $\frac{P(v_i)}{P(v_{i-1})}$ is unchanged since $P(v_i)$ and $P(v_{i-1})$ are changed by the same multiplicative factor. After $k$ steps we can then have $\frac{P(v_i)}{P(v_{i-1})} = \frac{R(v_i)}{R(v_{i-1})}$ for $1 < i \leq k$ and $P$ uniform on the remaining vectors. This allows one to calculate $P(v_i)$ explicitly in terms of $R$ and $\lambda$. Then calculating $KL(R||P)$ we see that $KL(R||P) \to 0$ as $\lambda \to \infty$. If $R$ is a distribution on $n$ nodes then there are at most $2^n$ hidden configurations so this construction adds at most $2^n$ hidden nodes. Furthermore one can add redundant hidden nodes to a Boltzmann machine with no effect on the distribution by setting its parameters to $0$, thus the stated result holds. ■

Smaller families of RBMs have been shown to be universal approximators but the construction used in Le Roux and Bengio will be useful later, in particular because the construction has all visible biases set to $0$. This allows one to "add" two RBMs together if they have the same number of visible nodes by simply including all hidden nodes with their respective

parameters of each RBM in a new RBM.

With these two examples one might wonder whether it is of interest to consider other networks except for the possibility of more efficient learning algorithms. One answer lies in the difficulty that stable networks have in modelling systems with time dependence. Although for any finite amount of time we can find a stable network to model such a system, in general, the number of hidden nodes required for this model depends on the number of time steps we want to simulate. We would like to be able to model a stochastic process more naturally, that is we would like a network to be able to continue to accurately model a process when the number of time steps increases without having to train it all over again, and for that we must turn to unstable networks. In doing so we need to come up with an alternate definition for universal approximation. Before discussing this we take a detour to mention the Turing completeness of neural networks.

## 3.2 Turing Completeness of Neural Networks

Viewing neural networks as computational models requires one to compare neural networks to more traditional machines, the most important one being the Turing machine. We say that something is Turing complete if it is capable of solving any problem a Turing machine can solve, or, to be more precise, capable of implementing the Turing-computable functions. Turing completeness of artificial neural networks was first conjectured at their conception [20]. An exact implementation of a Turing machine by an ANN was eventually given [5]. The notion of a computable function is a general one which does not make reference to any specific model of computation. A function is computable if it is computable by some computational model. The Church-Turing thesis states that the computable functions are exactly the Turing-computable functions. In our definition of neural networks we require there to be a finite number of nodes, but it has been shown that neural networks with an infinite number of hidden nodes are capable of *hypercomputation*. That is, they are able to implement non-Turing computable functions. This was done by showing that a neural network with an infinite number of hidden units can solve the halting problem [6].

## 3.3 Universal Approximation of Stochastic Processes

Although stable networks are good at approximating functions and distributions, often we are interested in systems with dynamic behaviour. We may try to use neural networks for tasks that have an inherent time dependence in which case the dynamics of stable networks are insufficient as they converge to a stationary distribution. In this case we must use unstable networks to achieve our goal. For unstable networks we would like to be able to say something about universal approximation that is similar to what we can say for stable networks. The most obvious definition would be as follows: given a stochastic process $\{R^{(t)}\}_{t=0}^{\infty}$ we want a class of networks such that for every $\epsilon > 0$, there exists a network in our class whose neural dynamics have the property $KL(R^{(t)}||P^{(t)}) < \epsilon$ for all $t \geq 0$. This definition has the disadvantage that it allows for the case that the KL-divergence of the infinite sequences diverges. A stronger condition would be to form the distributions $R$ and $P$ on infinite sequences and to say that a class is a universal approximator if for every $R$ formed in this way and each $\epsilon$, there is a distribution $P$ coming from the neural dynamics of some network in our class such that $KL(R||P) < \epsilon$. In practice this can be a difficult condition to satisfy. However, we may be able to take the first $k$ time steps of a stochastic process for arbitrary $k$ and approximate that distribution arbitrarily well by neural networks in our class. Thus we form the joint distribution of the stochastic process over the first $k$ steps denoted by $R_k$. This will be our definition for universal approximation of stochastic processes.

**Def:** *A class of Neural Networks with output nodes $V$ is a universal approximator for a stochastic process $\{R^{(i)}\}_{i=0}^{\infty}$ on $V$ if for each integer $k > 0$ and all $\epsilon > 0$ there exists a network with neural dynamics $\{P^{(t)}\}_{t=0}^{\infty}$ such that $KL(R_k||P_k) < \epsilon$.*

Note that the neural dynamics depend on a choice of initial distribution, and the definition requires only that the statement is satisfied for some initial distribution; however, in practice we may have a specific one in mind. As mentioned before we are really only interested in the case where the number of hidden nodes required by the network for a given approximation does not depend on the number of time steps for which we want to simulate $R$. We want our networks to approximate each $R_k$ uniformly with respect to the number of hidden nodes. In other words we are interested in a class of networks that is an $m$-family for some $M$. A class of neural networks is a universal approximator for a set of stochastic processes if it is a universal approximator for every process in

the set. Often we will be interested in approximating processes that all satisfy a certain condition. The most common requirement one could have on a stochastic process is that it satisfies the *Markov property*, that is it is time-homogeneous and has one step time dependence. In other words given any $k$, $R(v^{(0)},...,v^{(k)}) = \left(\prod_{i=1}^{k} R_1(v^{(i)}|v^{(i-1)})\right) R_0(v^{(0)})$. Here $R_1$ and $R_0$ are some distributions called the transition and initial distribution, respectively. One may weaken the requirement on the time dependence by allowing for any finite time dependence in which case we have that $R$ satisfies the property $R(v^T) = \left(\prod_{k=m}^{T-1} R_1(v^{(k)}|v^{(k-1)},...,v^{(k-m)})\right) R_0(v^{(0)},...,v^{(m-1)})$. If we want to find classes of networks that approximate the set of distributions with a certain finite time dependence in the the way defined above, one approach might be to factor the distribution $P$ into its marginals $P_t$ and compare each to either $R_1$ or $R_0$. To make sure this approach works we prove the following lemma.

**Lemma 3.2**: *Let $R$ be a distribution on a finite sequence of length $T$ on $n$-dimensional binary vectors that is time homogeneous with finite time dependence. Given a set of distributions $\mathbf{P}$ on the same sequences, if for every $\epsilon > 0$ we can find a distribution $P \in \mathbf{P}$ such that for every $v^T$, $KL(R_1(\cdot|v^{(t-1)},...,v^{(t-m)})||P_t(\cdot|v^{(t-1)},...,v^{(0)})) < \epsilon$ for $m \leq t < T-1$ and $KL(R_{0,t}(\cdot|v^{(t-1)},...,v^{(0)})||P_t(\cdot|v^{(t-1)},...,v^{(0)})) < \epsilon$ for $0 < t < m$ and $KL(R_{0,0}(\cdot)||P_0(\cdot)) < \epsilon$ then we can find distributions $P \in \mathbf{P}$ to approximate $R$ to arbitrary precision .*

*Proof*: For an arbitrary $\epsilon$, we need to find a $P \in \mathbf{P}$ such that $KL(R||P) < \epsilon$, writing the KL-divergence

$$KL(R||P) = \sum_{v^T} R(v^T) \log\left(\frac{R(v^T)}{P(v^T)}\right).$$

We can write $P(v^T)$ as

$$\left(\prod_{t=1}^{T-1} P(v^{(t)}|v^{(t-1)},...,v^{(0)})\right) P(v^{(0)})$$

and by assumption

$$R(v^T) = \left(\prod_{t=m}^{T-1} R_1(v^{(t)}|v^{(t-1)},...,v^{(t-m)})\right) \prod_{i=1}^{m-1} R_0(v^{(i)}|(v^{(i-1)},...,v^{(0)}) R_0(v^{(0)}).$$

Then expanding out the log in the KL-divergence gives us

$$KL(R||P) = \sum_{v^T} \sum_{t=m}^{T-1} R(v^T) \log \left( \frac{R_1(v^{(t)}|v^{(t-1)}, ..., v^{(t-m)})}{P(v^{(t)}|v^{(t-1)}, ..., v^{(0)})} \right)$$

$$+ \sum_{v^T} \sum_{t=1}^{m-1} R(v^T) \log \left( \frac{R_0(v^{(t)}|v^{(t-1)}, ..., v^{(0)})}{P(v^{(t)}|v^{(t-1)}, ..., v^{(0)})} \right) + \sum_{v^T} R(v^T) \log \left( \frac{R_0(v^{(0)}}{P(v^{(0)})} \right).$$

We can decompose $R(v^T)$ into $R(v^{(T-1)}, ..., v^{(t+1)}|v^{(t)})R(v^{(t)}|v^{(t-1)}, ..., v^{(0)})R(v^{(t-1)}, ..., v^{(0)})$. Note $R(v^{(T-1)}, ..., v^{(t+1)}|v^{(t)})R(v^{(t-1)}, ..., v^{(0)}) < 1$ so for a given $t$ we can write

$$\sum_{v^T} R(v^T) \log \left( \frac{R_1(v^{(t)}|v^{(t-1)}, ..., v^{(t-m)})}{P(v^{(t)}|v^{(t-1)}, ..., v^{(0)})} \right)$$

$$= \sum_{v^{(T-1)}, ..., v^{(t+1)}, v^{(t-1)}, ..., v^{(0)}} R(v^{(T-1)}, ..., v^{(t+1)}|v^{(t)})R(v^{(t-1)}, ..., v^{(0)})$$

$$\times \left( \sum_{v^{(t)}} R(v^{(t)}|v^{(t-1)}, ..., v^{(0)}) \log \left( \frac{R_0(v^{(t)}|v^{(t-1)}, ..., v^{(0)})}{P(v^{(t)}|v^{(t-1)}, ..., v^{(0)})} \right) \right).$$

The first factor is less than 1 for all $v^T$. $R_0(v^{(t)}|v^{(t-1)}, ..., v^{(0)}) = R_0(v^{(t)}|v^{(t-1)}, ..., v^{(t-m)})$ so the second factor is merely the KL-divergence of $R_1(\cdot|v^{(t-1)}, ..., v^{(t-m)})$ and $P_t(\cdot|v^{(t-1)}, ..., v^{(0)})$ which is positive. Finally $KL(R_{0,t}(\cdot|v^{(t-1)}, ..., v^{(0)})||P_t(\cdot|v^{(t-1)}, ..., v^{(0)}))$ is independent of vectors at time step $k > t$ giving us

$$\sum_{v^T} R(v^T) \log \left( \frac{R_1(v^{(t)}|v^{(t-1)}, ..., v^{(t-m)})}{P(v^{(t)}|v^{(t-1)}, ..., v^{(0)})} \right)$$

$$\leq 2^{(T-(t+1))n} \sum_{v^t} KL(R_1(\cdot|v^{(t-1)}, ..., v^{(t-m)})||P_t(\cdot|v^{(t-1)}, ..., v^{(0)})).$$

The same logic applies for the cases with $t < m$.

By hypothesis there exists $P \in \mathbf{P}$ such that for every $v^T$ and every $\epsilon'$,
$KL(R_1(\cdot|v^{(t-1)}, ..., v^{(t-m)})||P_t(\cdot|v^{(t-1)}, ..., v^{(0)})) < \epsilon'$ for $t \geq m$,
$KL(R_{0,t}(\cdot|v^{(t-1)}, ..., v^{(0)})||P_t(\cdot|v^{(t-1)}, ..., v^{(0)})) < \epsilon'$ for every $0 < t < m$, and
$KL(R_0||P_0) < \epsilon'$.
This gives us

$$KL(R||P)$$

$$\leq \sum_{t=m}^{T-1} 2^{(T-(t+1))n} \sum_{v^t} KL(R_1(\cdot|v^{(t-1)},...,v^{(t-m)})||P_t(\cdot|v^{(t-1)},...,v^{(0)}))$$

$$+ \sum_{t=1}^{m-1} 2^{(T-(t+1))n} \sum_{v^t} KL(R_0(\cdot|v^{(t-1)},...,v^{(0)})||P(\cdot|v^{(t-1)},...,v^{(0)}))$$

$$+2^{T-n} KL(R_0||P_0)$$

$$< \sum_{t=m}^{T-1} 2^{(T-1)n}\epsilon' + \sum_{t=0}^{m-1} 2^{(T-1)n}\epsilon'.$$

Then we merely choose an $\epsilon'$ so that this expression is less than $\epsilon$ and choose a corresponding $P \in \mathbf{P}$. ■

# Chapter 4

# Universal Approximation of Stochastic Processes by the TRBM and RTRBM

## 4.1 The TRBM

Although the Boltzmann machine is a universal approximator, in general there is no $k$-family of Boltzmann machines that is a universal approximator for a stochastic process since the Boltzmann machine is stable. Despite this the Boltzmann machine has several nice learning algorithms that we do not want to abandon. Thus, to deal with stochastic processes, we look for modifications of the Boltzmann machine that allow us to apply the learning rules we have. The key to the learning rules of the Boltzmann machine lies in performing gradient descent on the KL-divergence. Rather than start with a Boltzmann machine, which will eventually converge to a Boltzmann distribution, we define a network by taking a Boltzmann distribution at each time step. This network is obviously stable since it is independently drawing samples from a constant distribution at each time step. We want to modify this machine to make it unstable, and the easiest way to do that is to make the parameters of the Boltzmann distribution at the current time step depend on the state of the network at the previous time step. This is how we define the Temporal Restricted Boltzmann Machine or TRBM. The TRBM is a class of neural networks with one-step time dependence defined as follows: $X = \{0,1\}^n = (V,H)$,

$$\Omega_{\theta,(v',h')}(v,h) = \frac{\exp(\frac{1}{2}\sum_{i,j} x_i w^{(0)}_{i,j} x_j + \sum_i b^{(0)}_i x_i + \sum_{i,j} h'_i w^{(1)}_{i,j} h_j)}{Z(h')}.$$

The restrictions on the weights matrix $w_{i,j}^{(0)}$ are the same as the restrictions on the weight matrix of the RBM, that is there are symmetric connections between visible and hidden units and no connections between hidden units or between visible units. We will thus treat $W^{(0)}$ as a matrix representing the connections between visible and hidden units. The expression $v^\top Wh$ will be shorthand for $\frac{1}{2}\sum_{i,j} v_i w_{i,j}^{(0)} h_j$. The biases will be partitioned into visible and hidden biases $(b, c)$. The matrix $W^{(1)}$ contains no connections from visible nodes or to visible nodes and can be thought of as a matrix $W'$ of connections between hidden nodes. Thus we will use $h'^\top W'h$ as shorthand for $\sum_{i,j} h'_i w_{i,j}^{(1)} h_j$. Note there are no restrictions on the parameters $w_{i,j}^{(1)}$. $Z(h')$ is the normalizing factor of this distribution. We will consider neural dynamics in which the initial distribution $P_0$ is taken as the Boltzmann distribution with identical parameters except for the hidden biases which are taken as an additional vector $b_{init}$. Our current goal is to show that the TRBM is a universal approximator for Markov chains. As discussed in the previous section we not only want universal approximation but we want universal approximation by a $k$-family for some fixed $k$. To do this we fix a number of time steps $T$ and show that for any Markov chain $\{R^{(t)}\}_{t=0}^{\infty}$ and any $\epsilon > 0$ there is a TRBM with neural dynamics $\{P^{(t)}\}_{t=0}^{\infty}$ (in particular we take $P_0$ to be as described above) such that $KL(R_T||P_T) < \epsilon$. We then show that the number of hidden nodes required to make this approximation is bounded by some function of the number of visible nodes and independent of $T$. For simplicity we will refer to the distributions $R_T$ and $P_T$ merely as $R$ and $P$. The statement of our main theorem is then as follows

**Theorem 4.1**: *There is a $k$-family of TRBMs on $n$ visible nodes that is a universal approximator for the set of Markov chains on $n$ visible nodes.*

*Proof:* By the previous lemma we will be looking for a TRBM that can approximate the transition probabilities of $R$ along with its initial distribution. The proof will rely on the universal approximation properties of RBMs. The idea is that given one of the $2^n$ configurations of the visible units, $v$, there is an RBM with distribution $P_v$ approximating $R_1(\cdot|v^{(t-1)} = v)$ to a certain precision. The universal approximation results for RBMs tell us that this approximation can be made arbitrarily precise for an RBM with enough hidden units. Furthermore, this approximation can be done with visible biases set to $0$ [24]. We thus set all visible biases of our TRBM to $0$ and include each of the approximating RBMs without difficulty. We label these RBMs $H_1, ..., H_{2^n}$. Given a specific configuration of the visible nodes $v$, $H_v$ refers to the RBM chosen to approximate $R_1(\cdot|v^{(t-1)} = v)$. In other

words, $H_v$ is the set $H_i$ where $v$ is the $i^{th}$ configuration of the control nodes.

The challenge then is to signal the TRBM which of the $2^n$ RBMs should be active at the current time step. To do this we include $2^n$ additional hidden nodes which we will call *control nodes*, each corresponding to a particular configuration of the visible units. Thus we add $2^n$ control nodes, $h_{c,1}, ..., h_{c,2^n}$ corresponding to the hidden nodes $H_1, ..., H_{2^n}$. Again, given a particular visible configuration $v$, we denote the corresponding control node by $h_{c,v}$. The set of all control nodes will be denoted $H_c$. Note that $(c, v)$ is the label of $h_{c,v}$ and weights connecting to $h_{c,v}$ will be denoted $w_{(c,v),i}$ or $w'_{(c,v),j}$. The control nodes will signal which of the $H_i$'s should be active at the next time step. To accomplish this we will choose parameters such that when $v$ is observed at time $t-1$, $h_{c,v}$ will be on at time $t-1$ with a probability close to 1 and every other control node will be off at time $t-1$ with probability close to 1. For convenience this configuration of control nodes will be referred to as $H_{c,v} = 1$ and other configurations $H_{c,v} \neq 1$. Each $h_{c,v}$ will have strong negative temporal connections to every $H_{v'}$ with $v' \neq v$, in essence turning off every RBM corresponding to $R_1(\cdot|v^{(t-1)} = v')$, and leaving the RBM corresponding to $R_1(\cdot|v^{(t-1)} = v)$ active (See Fig. 4.1). We will break the proof down into four parts each corresponding to a condition to be satisfied. We must choose parameters so that each of these conditions are satisfied simultaneously. We then show that by satisfying these conditions the statement of the theorem holds.

First, we must be able to choose parameters so that given $v^{(t-1)} = v$, the probability that $H_{c,v}^{(t-1)} = 1$ can be made arbitrarily close to 1. Second, we must have that the control nodes have no impact on the visible distribution at the same time step so the transition probabilities of the TRBM will still approximate $R_1(\cdot|v^{(t-1)} = v)$. Third, we must be able to choose parameters so that given $H_{c,v}^{(t-1)} = 1$, the probability that any nodes in $H_{v'}$ are on at time $t$ can be made arbitrarily close to 0 for $v' \neq v$. Finally, we must be able to approximate the initial distribution $R_0$.

To choose temporal connections, define $w'_{(c,v),j}$ to be $-\alpha$ if $h_j \in H_{v'}$ where $v' \neq v$ and 0 otherwise. Let every other $w'_{i,j} = 0$. In particular, remembering that $W'$ is not necessarily symmetric, we have $w'_{j,(c,v)} = 0$ for all $h_j$. The only parameters left to define are the biases and visible-hidden connections of $H_c$. Let $b_{(c,v)} = -(k - 0.5)\beta$ where $k$ is the number of visible nodes on in $v$. Finally, define the connections from the visible units to $h_{c,v}$ by $w_{i,(c,v)} = \beta$ if $v_i$ is on in the configuration $v$ and $-\beta$ otherwise. Here the parameters

of the control nodes are completely determined by $\alpha$ and $\beta$. We will proceed by showing that all necessary conditions are satisfied when $\alpha$ and $\beta$ are large enough.

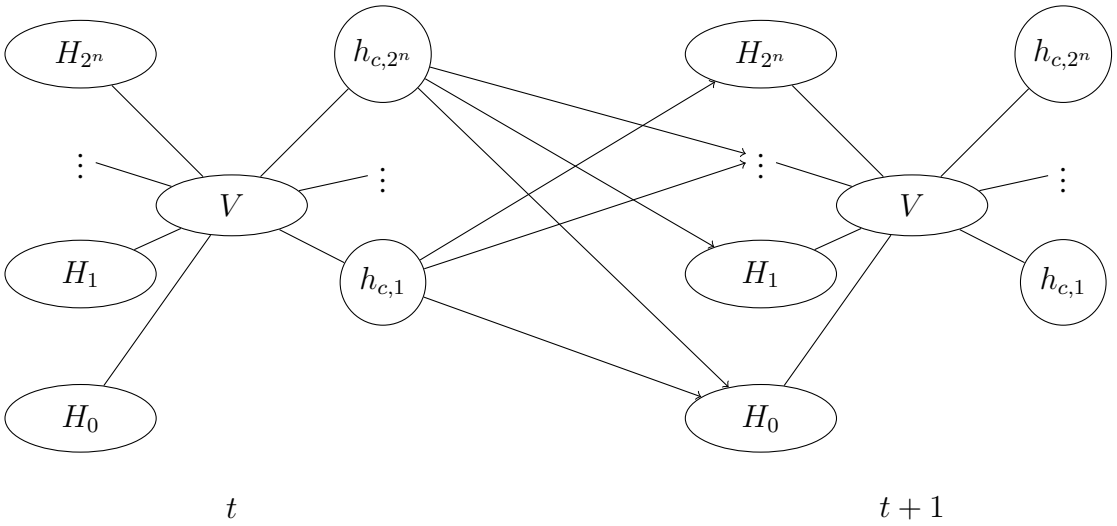


Figure 4.1: *Interactions between sets of nodes within and between time steps: each $h_{c,v}$ turns off every $H_{v'} : v \neq v'$ in the subsequent time step. $h_{c,v}$ will only be on if $v$ is observed at time $t$ and collectively the control nodes $H_c$ have negligible effect on the visible distribution. $H_0$ is an additional set of hidden units that will be used to model the initial distribution.*

Step 1:

For this step we show that as $\beta \to \infty$ we have $P(H^{(t)}_{c,v^{(t)}} = 1|v^{(t)}, \ldots, v^{(0)}) \to 1$. Note that given the visible state at time $t$, the state of the control nodes at time $t$ is conditionally independent of all other previous states. With this in mind, we can write the probability of a control node being on at time $t$ as

$$P(h^{(t)}_{c,v} = 1|v^{(t)}, ..., v^{(0)}) = \sigma(\sum_i v^{(t)}_i w_{i,(c,v)} + b_{(c,v)}),$$

where $\sigma$ is the logistic function. Note that for all $v^{(t)} \in \{0,1\}^n$ and $v \in \{0,1\}^n$, we have $\sum_i v^{(t)}_i w_{i,(c,v)} = a\beta - b\beta$ where $a$ is the number of nodes on in $v$ and $v^{(t)}$ and $b$ is the number of nodes on in $v^{(t)}$ but off in $v$. Since $b_{(c,v)} = -(k-0.5)\beta$ if $v \neq v^{(t)}$, then either $a < k$, in which case $\sum_i v^{(t)}_i w_{i,(c,v)} + b_{(c,v)} \leq -0.5\beta$, or $a = k$ and $b \geq 1$, which again implies $\sum_i v^{(t)}_i w_{i,(c,v)} + b_{(c,v)} \leq -0.5\beta$. If $v = v^{(t)}$ then $a\beta - b\beta + b_{(c,v)} = n\beta - (k-0.5)\beta = 0.5\beta$. Thus if $v = v^{(t)}$,

$$\sigma(\sum_i v^{(t)}_i w_{i,(c,v)} + b_{(c,v)}) = \sigma(0.5\beta).$$

Otherwise

$$\sigma(\sum_i v_i^{(t)} w_{i,(c,v)} + b_{(c,v)}) \leq \sigma(-0.5\beta).$$

So as $\beta \to \infty$, $P(H^{(t)}_{c,v^{(t)}} = 1 | v^{(t)}, ..., v^{(0)}) \to 1$.

Step 2:

Here we show that by making $\beta$ large enough we can make the effect of the control nodes on the visible distribution negligible. We have

$$P(v^{(t)}|h^{(t-1)}) = \frac{\sum_{h^{(t)}} \exp\left(\sum_{i,j} v_i^{(t)} h_j^{(t)} w_{i,j} + \sum_j b_j h_j^{(t)} + \sum_{i,j} h_i^{(t)} h_{i-1}^{(t-1)} w'_{i,j}\right)}{\sum_{v^{(t)}} \sum_{h^{(t)}} \exp\left(\sum_{i,j} v_i^{(t)} h_j^{(t)} w_{i,j} + \sum_j b_j h_j^{(t)} + \sum_{i,j} h_i^{(t)} h_{i-1}^{(t-1)} w'_{i,j}\right)}.$$

In the previous step we showed that by taking $\beta$ large enough, $P(H^{(t)}_{c,v^{(t)}} = 1 | v^{(t)}, ..., v^{(0)})$ becomes arbitrarily close to 1. So in the limit we have for all $v^{(t)}$

$$P(v^{(t)}|h^{(t-1)}) =$$

$$\frac{\sum_{h^{(t)}: H_{c,v^{(t)}}=1} \exp\left(\sum_{i,j: h_j \in (H \setminus H_c)} v_i^{(t)} h_j^{(t)} w_{i,j} + \sum_{j: h_j \in (H \setminus H_c)} b_j h_j^{(t)} + 0.5\beta + \sum_{i,j} h_i^{(t)} h_{i-1}^{(t-1)} w'_{i,j}\right)}{\sum_{v^{(t)}} \sum_{h^{(t)}: H_{c,v^{(t)}}=1} \exp\left(\sum_{i,j: h_j \in (H \setminus H_c)} v_i^{(t)} h_j^{(t)} w_{i,j} + \sum_{j: h_j \in (H \setminus H_c)} b_j h_j^{(t)} + 0.5\beta + \sum_{i,j} h_i^{(t)} h_{i-1}^{(t-1)} w'_{i,j}\right)}$$

$$= \frac{\sum_{h^{(t)}: H_{c,v^{(t)}}=1} \exp\left(\sum_{i,j: h_j \in (H \setminus H_c)} v_i^{(t)} h_j^{(t)} w_{i,j} + \sum_{j: h_j \in (H \setminus H_c)} b_j h_j^{(t)} + \sum_{i,j} h_i^{(t)} h_{i-1}^{(t-1)} w'_{i,j}\right)}{\sum_{v^{(t)}} \sum_{h^{(t)}: H_{c,v^{(t)}}=1} \exp\left(\sum_{i,j: h_j \in (H \setminus H_c)} v_i^{(t)} h_j^{(t)} w_{i,j} + \sum_{j: h_j \in (H \setminus H_c)} b_j h_j^{(t)} + \sum_{i,j} h_i^{(t)} h_{i-1}^{(t-1)} w'_{i,j}\right)}.$$

But this is just the distribution of the machine without the control nodes. Thus by making $\beta$ large enough, the effect of the control nodes on the visible distribution at the current time step becomes negligible.

Step 3:

In this step, remembering that $P_v$ is the distribution of the RBM corresponding to the distribution $R_1(\cdot|v^{(t-1)} = v)$, we show that as $\alpha$ and $\beta$ are increased to infinity, if $H_{c,v}^{(t-1)} = 1$ then $P(v^{(t)}|h^{(t-1)}) \to P_v(v^{(t)})$ for all $v^{(t)}$. Consider some configuration $h^{(t-1)}$ with $H_{c,v}^{(t-1)} = 1$. Take a hidden unit $h_j \in H_{v'}$ with $v' \neq v$. Since $v' \neq v$ and $H_{c,v}^{(t-1)} = 1$, then $h_{c,v}^{(t-1)} = 1$ and $w_{(c,v),j} = -\alpha$, so we have for all $v^{(t)}$

$$P(h_j^{(t)} = 1|v^{(t)}, h^{(t-1)}) = \sigma(\sum_i v_i^{(t)} w_{i,j} + b_j - \alpha).$$

Since $h_j$ is not a control node, $w_{i,j}$ is fixed for all $v_i$. Thus as $\alpha \to \infty$, $P(h_j^{(t)} = 1|h^{(t-1)}) \to 0$ for $h_j \in H_{v'}$ with $v' \neq v$. As $\beta \to \infty$ the effect of the control nodes on the visible distribution at time $t$ vanishes. Thus in the limit as $\alpha, \beta \to \infty$ we can ignore the effect of the control nodes, and the probability that any hidden node in $H_{v'}$ is on goes to 0. This leaves us with

$$P(v^{(t)}|h^{(t-1)}) \to \frac{\sum\limits_{H_v^{(t)}} \exp\left(\sum\limits_{i,j:h_j \in H_v} v_i^{(t)} h_j^{(t)} w_{i,j} + \sum\limits_{j:h_j \in H_v} b_j h_j^{(t)}\right)}{\sum\limits_{v^{(t)}} \sum\limits_{H_v^{(t)}} \exp\left(\sum\limits_{i,j:h_j \in H_v} v_i^{(t)} h_j^{(t)} w_{i,j} + \sum\limits_{j:h_j \in H_v} b_j h_j^{(t)}\right)} = P_v(v^{(t)}).$$

Here $H_v^{(t)}$ is a configuration of $H_v$.

Step 4:
Finally, we must also be able to approximate the initial distribution $R_0$ to arbitrary precision. We know there is an RBM $H_0$ with visible biases 0 whose Boltzmann distribution can approximate $R_0$ to a certain, arbitrary, precision. Include this machine in our TRBM. Now we define the initial biases. Let $b_{i,init} = \gamma$ for every $h_i \in H_0$ and $b_{c,v,init} = 0$ for $(c, v)$. Set $b_{i,init} = -\gamma$ for all other hidden nodes. Add $-\gamma$ to the biases of $H_0$. Regardless of the parameters chosen for any hidden node in some $H_v$, by making $\gamma$ large, every hidden node not in $H_0$ or $H_c$ will be turned off with probability close to 1 in the first time step. By making $\beta$ large Step 1 tells us that the effect of $H_c$ on the initial distribution is made arbitrarily small so $P_0$ can be made arbitrarily close to the Boltzmann distribution of $H_0$, which by construction approximates $R_0$. At subsequent time steps, by making $\gamma$ large, every node in $H_0$ will be off with probability close to 1, leaving us with the machine described in the first three steps. Note that this construction allows the control nodes to be active in the first time step and to transmit temporal data without disturbing the initial distribution.

Now we put the four steps together. Given an arbitrary $0 < t < T$, we can write each $P(v^{(t)}|v^{(t-1)}, ..., v^{(0)})$ as

$$\sum_{h^{(t-1)}} P(v^{(t)}, h^{(t-1)}|v^{(t-1)}, ..., v^{(0)})$$

$$= \sum_{h^{(t-1)}} P(v^{(t)}|h^{(t-1)}, v^{(t-1)}, ..., v^{(0)})P(h^{(t-1)}|v^{(t-1)}, ..., v^{(0)})$$

$$= \sum_{h^{(t-1)}} P(v^{(t)}|h^{(t-1)})P(h^{(t-1)}|v^{(t-1)}, ..., v^{(0)}).$$

Step 1 tells us that as $\beta \to \infty$

$$\sum_{h^{(t-1)}} P(v^{(t)}|h^{(t-1)})P(h^{(t-1)}|v^{(t-1)}, ..., v^{(0)})$$

$$\to \sum_{h^{(t-1)}, H^{(t-1)}_{c,v^{(t-1)}}=1} P(v^{(t)}|h^{(t-1)})P(h^{(t-1)}|v^{(t-1)}, ..., v^{(0)})$$

for all $v^{(0)}, ..., v^{(t-1)}$. Steps 2 and 3 tell us that as $\alpha, \beta \to \infty$ we get that $P(v^{(t)}|h^{(t-1)}) \to P_{v^{(t-1)}}(v^{(t)})$ for all $h^{(t-1)}$ with $H^{(t-1)}_{c,v^{(t-1)}} = 1$. Using the fact that as $\beta$ goes to infinity, $\sum_{h^{(t-1)}, H_{c,v^{t-1)}}=1} P(h^{(t-1)}|v^{(t-1)}, ..., v^{(0)}) \to 1$, we have that $\sum_{h^{(t-1)}} P(v^{(t)}|h^{(t-1)})P(h^{(t-1)}|v^{(t-1)}, ..., v^{(0)})$ can be made arbitrarily close to $P_{v^{(t-1)}}(v^{(t)})$ for all $v^{(0)}, ..., v^{(t)}$ in terms of Euclidean distance. By construction, $KL(R(\cdot|v^{(t-1)})||P_{v^{(t-1)}}) < \epsilon'$ for some arbitrary $\epsilon' > 0$. Taking the limit as $\alpha, \beta \to \infty$ we get that $KL(R(\cdot|v^{(t-1)})||P(\cdot|v^{(t-1)}, ..., v^{(0)})) = KL(R(\cdot|v^{(t-1)})||P_{v^{(t-1)}}) < \epsilon'$. Finally Step 4 tells us that in the limit as $\gamma \to \infty$, $KL(R_0||P_0) < \epsilon'$. So by Lemma 3.2 the result holds. ■

## 4.2 The Generalized TRBM

The TRBM used in the previous section is a restricted instance of a more generalized model described by Sutskever et al. (2006). In the full model we allow explicit long-term hidden-hidden connections as well as long-term visible-visible connections. In this paper we will not consider models with visible-visible temporal interaction. The generalized TRBM is

given by

$$\Omega_{\theta,v^{(t-1)},...,v^{(t-m)},h^{(t-1)},...,h^{(t-m)}}(v^{(t)},h^{(t)})$$
$$= \frac{\exp(v^{(t)\top}W^{(0)}h^{(t)} + c^\top v^{(t)} + b^\top h^{(t)} + h^{(t)\top}W^{(1)}h^{(t-1)} + ... + h^{(t)\top}W^{(m)}h^{(t-m)})}{Z(h^{(t-1)},...,h^{(t-m)})}.$$

This is a slight abuse of notation as the matricies $W^{(i)}$ represent connections between the entire set of neurons rather than just hidden to hidden or visible to visible. However, as before, we have $w^{(0)}_{i,j} = 0$ if $x_i$ and $x_j$ are both in $V$ or $H$ and $w^{(i)}_{i,j} = 0$ if either $x_i$ or $x_j$ is in $V$. For $t < m$ we define the distribution in the following way: for each $l$ with $l > t$, we define $\Omega_{\theta,v^{(t-1)},...,v^{(0)},h^{(t-1)},...,h^{(0)}}(v^{(t)},h^{(t)})$ by replacing each corresponding $W^{(l)}h^{(t-l)}$ with the initial bias $b^{(l)}_{init}$.
If we drop the restriction that $R$ be a Markov chain we can generalize the previous theorem so that $R$ is any distribution homogeneous in time with a finite time dependence.

**Theorem 4.2:** *There is a $k$-family of generalized TRBMs on $n$ visible nodes that is a universal approximator for the set of stochastic processes with finite time dependence on $n$ visible nodes.*

*Proof:* The proof is similar to the proof of Theorem 4.1. Let $m$ be the time dependence of $R$. Then for each visible sequence $v^{(t-1)},...,v^{(t-m)}$ we construct a TRBM $P$ by adding sets of hidden units $H_{v^{(t-1)},...,v^{(t-m)}}$ with parameters chosen to approximate $R_1(\cdot|v^{(t-1)},...,v^{(t-m)})$. For each visible configuration $v$ we add a control unit $h_{c,v}$ with the same bias and visible-hidden connections (determined by a parameter $\beta$) as in the construction for Theorem 4.1. If $i \leq m$, define the $i$-step temporal connections as $w^{(i)}_{(c,v),j} = -\alpha$ if $h_j \in H_{v^{(t-1)},...,v^{(t-i)},...,v^{(t-m)}}$ with $v^{(t-i)} \neq v$ and 0 otherwise. All other temporal connections are set to 0. Then by repeating Step 1, Step 2, and Step 3 in the proof of Theorem 4.1, and by making $\alpha$ and $\beta$ sufficiently large we can make

$$KL(R_1(\cdot|v^{(t-1)},...,v^{(t-m)})||P(\cdot|v^{(t-1)},...,v^{(0)}))$$

arbitrarily small for all $v^T$.

To finish the proof we must modify the machine to approximate the $m$ initial distributions as well. To do this we add sets of hidden units $H_{(i)}$ for each $0 \leq i < m$ with parameters that approximate $R_{0,i}(\cdot|v^{(i-1)},...,v^{(0)})$ to a certain precision for all sequences $v^i$. We know that this can be done using the construction in the previous step. Now we want every non-

control node in $H_{(i)}$ to be inactive for all time except $t = i$. To do this we will give every non-control node in $H_{(i)}$ a strong negative bias. We add a single additional control node $h_c$ with $i$-step temporal connections to counteract the bias added to $H_{(i)}$. We thus add $-\gamma$ to the biases in $H_{(i)}$ excluding the control nodes of $H_{(i)}$. Define the bias of $h_c$ to be $-\gamma$ and the $i$-step temporal connections to be $\gamma$ for each node in $H_{(i)}$ excluding the control nodes, and $-\gamma$ for every other non-control node. Set $b^{(1)}_{i,init}$ to $-\gamma$ for every non-control hidden unit that is not in $H_0$ and let $b^{(k)}_{init}$ be 0 for all $k > 0$. Let $b^{(1)}_{c,init}$ be $2\gamma$ and $b^{(1)}_{i,init}$ be 0 for the remaining nodes ($H_0$ and control nodes). Let $h_c$ have no connections to visible units. By making $\gamma$ large, $h_c$ will be on with probability close to 1 and every non-control hidden unit not in $H_0$ will be off with probability close to 1 in the first time step. In subsequent time steps, $i < m$, $h_c$ will be off with probability close to 1 and the temporal connections from $h_c$ will be $-\gamma$ to nodes in $H_{v^{(t-1)},...,v^{(t-m)}}$ for all $v^{(t-1)}, ..., v^{(t-m)}$ and $\gamma$ to nodes in $H_{(i)}$. By making $\gamma$ large, the temporal connections turn off every non-control node in $H_{v^{(t-1)},...,v^{(t-m)}}$ and the biases turn off every non-control node in $H_{(k)}$ with $k \neq i$. At time $i$, the $i$-step temporal connections from $h_c$ to $H_{(i)}$ cancel the effect of the bias terms leaving $H_{(i)}$ the only set of nodes active at time $i$. Thus, at each time step $0 \leq i < m$, $P(\cdot|v^{(i)}, ..., v^{(0)})$ can be made arbitrarily close to the distribution given by $H_i$. For $t \geq m$, $h_c$ and $H_{(i)}$ will be off with probability close to 1 for all $i$, leaving us with the machine described in the first step. ■

## 4.3 The Recurrent Temporal Restricted Boltzmann Machines

The TRBM gives a nice way of using a Boltzmann Machine to define a probability distribution that captures time dependence in data, but it turns out to be difficult to train in practice [30]. To fix this, a slight variation of the model, the RTRBM, was introduced. The key difference between the TRBM and the RTRBM is the use of deterministic real valued parameters denoted $h_t$. We will denote the probabilistic binary hidden units at time $t$ by $h'^{(t)}$. The distribution defined by an RTRBM, $Q$, is

$$\Omega_{\theta,v^{(t-1)},h^{(t-1)}}(v^{(t)}, h^{(t)}) = \frac{\exp(v^{(t)\top} W h'^{(t)} + c^{(0)\top v^{(t)}} + b^{(0)\top} h'^{(t)} + h'^{(t)\top} W' h^{(t-1)})}{Z(h^{(t-1)})},$$

where $W$ and $W'$ are defined in the same way as the TRBM and $h^{(t)}$ is defined by

$$\begin{aligned} h^{(t)} &= \sigma(Wv^{(t)} + W'h^{(t-1)} + b), \\ h^{(0)} &= \sigma(Wv^{(0)} + b_{init} + b), \end{aligned} \tag{4.1}$$

where $\sigma$ is the logistic function and $b_{init}$ is an initial bias. $P_0$ is once again defined as a Boltzmann Distribution with bias $b + b_{init}$. The difference between the RTRBM and the TRBM is the use of the sequence of real valued vectors $h^T$ for the temporal connections. At each time step each hidden node $h_i$ takes on two values, a deterministic $h_i^{(t)}$ and a probabilistic $h_i^{'(t)}$. The fact that the temporal parameters are calculated deterministically makes learning more tractable in these machines [30].

**Theorem 4.3**: *Let R be a distribution over a sequence of length T of binary vectors of length n that is time homogeneous and has finite time dependence. For all $\epsilon > 0$ there exists an RTRBM, Q, such that $KL(R||Q) < \epsilon$.*

*Proof:* As in the proof of Theorem 4.1, for each configuration $v^{(t-1)}, ..., v^{(t-m)}$, include hidden units $H_{v^{(t-1)},...,v^{(t-m)}}$ with parameters so that the KL distance between the visible distribution of the Boltzmann machine given by $H_{v^{(t-1)},...,v^{(t-m)}}$ with these parameters and $R_1(\cdot|v^{(t-1)}, ..., v^{(t-m)})$ is less than $\epsilon'$. Now for each possible visible configuration $v$ add the control node $h_{c,v}$ with the same biases and visible-hidden weights as in the proofs of Theorems 4.1 and 4.2 (determined entirely by parameter $\beta$). In the proof of Theorem 4.2, $h_{c,v}$ had $i$-step temporal connections from $h_{c,v}$ to every $H_{v^{(t-1)},...,v^{(t-m)}}$ with $v^{(t-i)} \neq v$. The proof will proceed by showing that each of these $i$-step temporal connections can instead by given by a chain of nodes in the RTRBM. We wish to show that we can add $i$ hidden units connecting $h_{c,v}$ to every hidden node in $H_{v^{(t-1)},...,v^{(t-m)}}$ such that if $h_{c,v}$ is on at time $t - i$, it will have the same effect on the distribution of a node in $H_{v^{(t-1)},...,v^{(t-m)}}$ as it does in the general TRBM, and the $i$ additional hidden units do not effect the visible distribution. If we can achieve this then the same proof will hold for the RTRBM. This will be done as follows.

For each $h_{c,v}$ and each $1 \leq k < m$, add an additional hidden unit with 0 bias and no visible-hidden connections. For each $h_{c,v}$, label these $m - 1$ hidden units $g_{(v,1)}, ..., g_{(v,m-1)}$. Since these nodes have no visible-hidden connections they have no effect on the visible distribution at the current time step. For $1 < k < m - 1$, let $w'_{(v,k),(v,k+1)} = 1$. Let $w'_{(c,v),(v,1)} = 1$ and $w'_{(v,k),j} = -\alpha$ if $h_j \in H_{v^{(t-1)},...,v^{(t-k-1)},...,v^{(t-m)}}$ with $v^{(t-k-1)} \neq v$, and $w'_{(v,k),j} = 0$

otherwise (see Fig. 4.2). Given a sequence $v^{(t-1)}, ..., v^{(t-m)}$, consider the probability that $h_j^{'(t)} = 1$ for some hidden unit $h_j \in H_{v^{(t-1)'},...,v^{(t-m)'}}$ where $v^{(t-k)'} \neq v^{(t-k)}$ for some $k$. Then by construction there is some hidden node $g_{k-1}$ with $w'_{(v,k-1),j} = -\alpha$. The value of $g_{k-1}^{(t-1)}$ is calculated recursively by $g_{k-1}^{(t-1)} = \sigma(g_{k-2}^{(t-2)}) = \sigma^{k-1}(h_{c,v^{(t-k)}}^{(t-k)}) = \sigma^k(0.5\beta)$. Since $k$ is bounded, by making $\beta$ arbitrarily large we make $g_{k-1}$ arbitrarily close to 1 and thus make $h_j^{(t)'} w'_{(v,k-1),j} g_{k-1}^{(t-1)}$ arbitrarily close to $-\alpha$.

Now suppose we have $h_j^{(t)'} = 1$ for some hidden unit in $H_{v^{(t-1)},...,v^{(t-m)}}$. Then every temporal connection is $-\alpha$, and $g_{(v,k)}^{(t-1)} = \sigma^k(-0.5\beta)$ for every $g_{(v,k)}$, so again by making $\beta$ arbitrarily large we make the temporal terms arbitrarily close to 0. Thus as $\beta \to \infty$, the temporal terms from $h_{c,v}^{(t-i)}$ are either $-\alpha$ or 0 as needed.

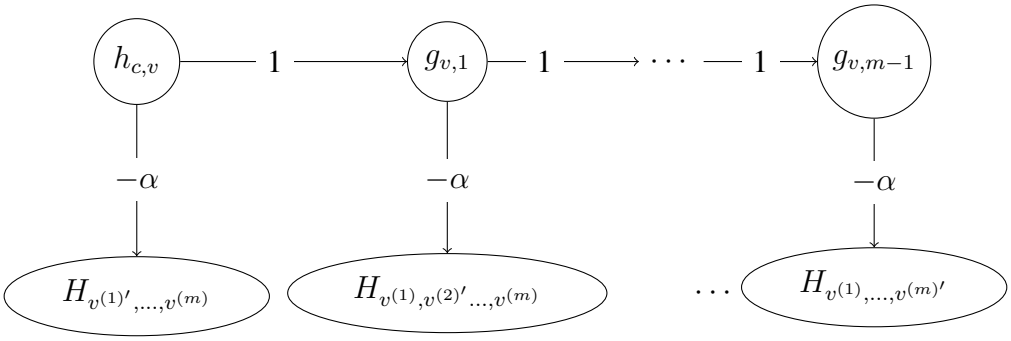


Figure 4.2: *The temporal connections of a control node. Each $v^{(i)'}$ is a configuration not equal to $v$*

To finish the proof, note that our machine must also be able to approximate the initial $m$ distributions. As before we will use the construction of Theorem 4.2 and replace the long-term temporal connections with a chain. To begin, add each $H_{(i)}$ described in the proof of Theorem 4.2. Add hidden units $l_1, ..., l_{m-1}$ along with $h_c$ where the temporal connections have weight $2\gamma$ from $l_i$ to every non-control node in $H_{(i)}$ and $-2\gamma$ to every non-control node in $H_{v^{(t)},...,v^{(t-m)}}$ for all $v^{(t)}, ..., v^{(t-m)}$. Now set the initial biases for every $l_i$ to be 0 except for $l_1$. Set this initial bias to be $2\delta$. Define the initial bias for every other non-control node to be $-\delta$ with the exception of $H_0$ whose initial bias is 0 (see Fig. 4.3). Now for the first time step, by making $\delta$ large we make the initial distribution arbitrarily close to $H_{(0)}$ allowing us to approximate the distribution for the first time step. Now consider $l_i^{(i)}$ for $0 < i \leq m$. By equation (1), $l_2^{(2)} = \sigma(\delta\sigma(\delta) - \delta)$. By making $\delta$ large we make $l_2^{(2)} \approx 0.5$. Now $l_3^{(3)} \approx \sigma(0.5\delta - 0.5\delta) = 0.5$. Then by induction $l_i^{(i)} = 0.5$. For every other $l_j^{(i)}$ we have $l_j^{(i)} \approx \sigma(-0.5\delta)$. By making $\delta$ large we have each $l_j^{(i)} \approx 0$ to arbitrary precision for

$i \neq j$. The temporal terms given by $l_i^{(i)}$ are then equal to $-\gamma$ for nodes in $H_{v^{(t)},\ldots,v^{(t-m)}}$ and $\gamma$ for nodes in $H_{(i)}$ making the construction behave the same way as that in Theorem 4.3. ■

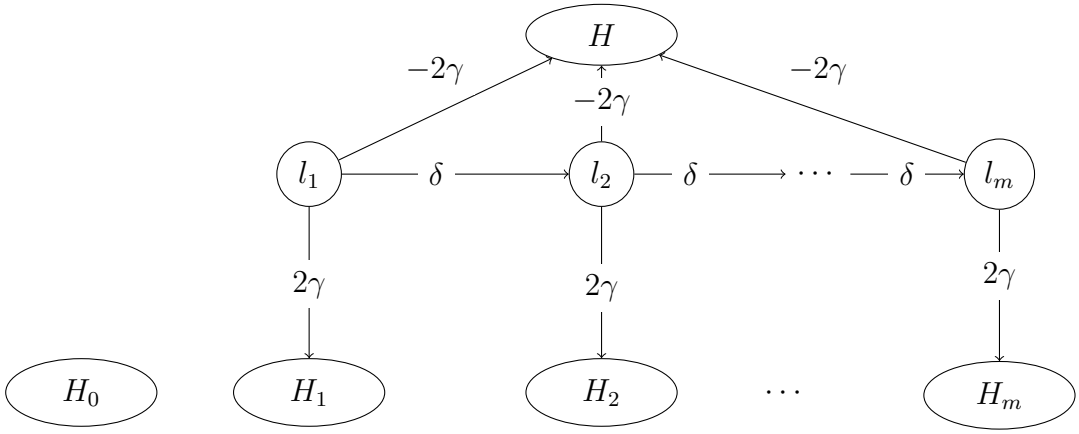


Figure 4.3: *The initial chain. $H$ is the machine defined in the first part of the proof. Each connection from an $l_i$ to $H$ or $H_k$ only goes to non-control nodes in the set (control nodes include the chains $g_{(v,1)}, ..., g_{(v,m)}$). The $l_i$'s have no visible connections.*

# Chapter 5

# Experimental Results

## 5.1 Experimental Results

Signals in the human brain are propagated via neurotransmitters. Each neurotransmitter has a variety of receptors that it binds to and some it does not bind to. The human brain contains a large number of different neurotransmitters and their corresponding receptors. This gives rise to the view of the brain as a series of interacting signalling pathways each given by a specific neurotransmitter. Selective alterations in the function of a specific neurotransmitter can have drastic effects on both perception and cognition. Imbalances in levels of neurotransmitters in the brain have been implicated in a wide variety of mental disorders from depression to schizophrenia [27][16]. The computational power of ANNs (and in particualr RTRBMs) raises the question of what kind of experiments can be done on ANNs to provide insight into the sensory aspects of these disorders.

Varying effectiveness of neurotransmitters can be seen as a perturbation in the signalling pathways of the brain. We would like to recreate this type of perturbation using ANNs. The most obvious way to replicate this effect in an ANN is by scaling sets of parameters by some multiplicative factor. Although one can partition the parameters of an ANN arbitrarily, this procedure would presumably be more effective when parameters are grouped by functional significance. As discussed before, in most ANNs the weights are functionally equivalent before training. After training, one may utilize methods discussed in the previous section to identify distinct functional groups of weights, for example, edge detectors.

Some neural networks have different sets of parameters with pre-defined roles. That makes

the process of parameter scaling simpler as the network already comes equipped with distinct sets of parameters. The RTRBM comes pre-defined with two distinct sets of parameters, those used to communicate within a time step and those used to communicate between time steps.

## 5.2 The Reconstructed Distribution

For trained probablistic networks we are able to directly observe how well the network models the training data by simply sampling from the distribution comparing the visible units to examples from the training data. In a trained network with hidden nodes, we can view the hidden nodes as encoding some information about the visible distribution. This can be seen as an encoding/decoding procedure. Given a visible sequence we sample from the conditional distribution of the hidden nodes to produce a corresponding hidden sequence. From that point we can "decode" the sequence by sampling from the conditional distribution of the visible units. In the case we are trying to approximate a certain distribution this procedure forms what we call the reconstructed distribution. Formally, given a neural network with hidden nodes $H$ whose neural dynamics have marginal distribution over the first $k$ time steps $P_k$ and target distribution $R_k$ we define the reconstructed distribution as $P_{(r,k)}(v^k) = \sum_{h^k, v_0^k} P_k(v^k|h^k)P(h^k|v_0^k)R_k(v_0^k)$ (see Fig. 5.1).

Given the similarities in the hierarchical nature of the deep belief network and the visual cortex one might wonder what experiments on the Boltzmann machine can tell us about the visual system. The analogy between the two becomes even stronger in the case of the temporal Boltzmann machine. Tweaking the parameters of a trained (R)TRBM will lead to distinct qualitative changes in the reconstructed distribution. By changing the weights and observing the corresponding changes in the reconstructed distribution, one can get an idea how those weights are used by the machine to relay information. By changing sets of weights that are somehow similar one can get some indication of how those sets of weights are used. In the (R)TRBM there are two distinct kinds of weights, temporal and visible-hidden. This sets the stage for a type of *in-silico* experiment on perception. This is done by first training an (R)TRBM to model some form of sensory data, then scaling the temporal and visible-hidden connections via some multiplicative factors $\alpha$ and $\beta$, respectively, and finally by comparing the original data to the reconstructed distribution. The procedure of scaling related sets of weights can be seen as analogous to a shift in the connectivity of the cortex either via neurological or exogenous factors.

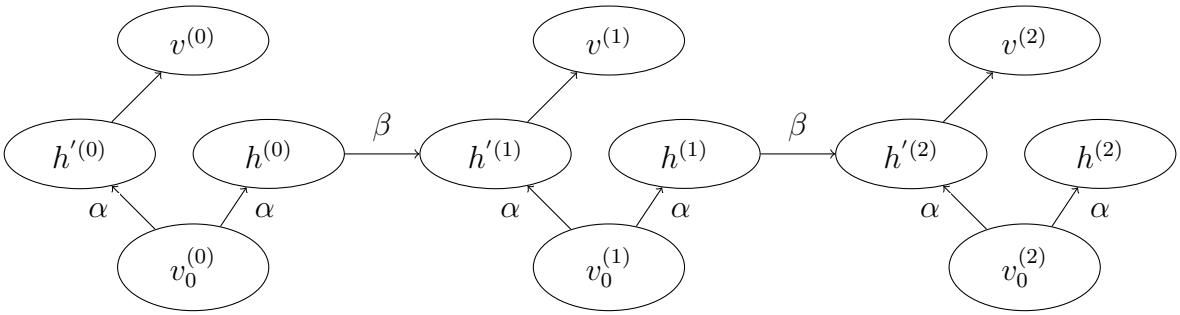


Figure 5.1: *The reconstruction process: Starting with a sequence of input vectors, $v_0^T$, the hidden sequence $h'^T$ is produced after scaling the parameters of the RTRBM by $\alpha$ and $\beta$. From there the reconstructed sequence, $v^T$, is produced either by sampling or mean-field approximation. Note that using a mean-field approximation to encode data in the hidden units amounts to using $h^T$ to reconstruct $v^T$.*

## 5.3 Simulations

Throughout this section $\alpha$ will refer to the scaling of the visible-hidden connections and $\beta$ will refer to the scaling of the hidden-hidden connections. The experimental results reported in Sutskever et al. (2008) were replicated; a single layer RTRBM with $400$ hidden nodes was trained for $100,000$ iterations on sequences of bouncing balls (represented as sequences of vectors with values ranging from $0$ to $1$) using gradient descent. The gradient was approximated using backpropagation through time along with contrastive divergence. When tasked with trying to predict the next frame, the trained network has a mean-squared error per visible unit per timestep of approximately $0.0066$ (using a mean-field approximation for the hidden units). This network was then given a sample sequence to reconstruct under various scaling factors ($(\alpha, \beta) = (1,1), (0.5,1), (0,1), (1,0.5), (1,0), (0.5,0.5)$). The machine was then trained on the motion capture (MOCAP) data found in [31]. MOCAP data consists of positional vectors corresponding to sensors attached to the human body. This data was real-valued data with $0$ mean and a variance of $1$ but with maximum $21.6$ and minimum $-14$. The machine was trained for $100,000$ iterations with $200$ hidden units, resulting in a mean-squared error per visible unit per timestep of approximately $0.429$ (again using a mean-field approximation). A two-layer RTRBM was then tested on the MOCAP data with 100 hidden units trained for 50000 iterations for both layers. The two-layer model was trained greedily using the binary activations of the first layer as input to the second. When evaluated empirically, the two-layer model performed marginally better using the same metric but with binary sampling instead of the mean-field approximation ($1.4$ for the two-layer compared to $1.43$ for the single-layer). With mean-field approximation the two-

layer architecture performed identically or even slightly worse using this metric. This is most likely due to the fact that the second layer was trained on binary samples of the hidden units rather than their mean-field approximation.

For the video data of bouncing balls, the reconstructions with scaled down temporal weights and unchanged visible weights show the balls very clearly but the position of the balls is erratic and in the extreme case doesn't correspond to the position of the balls in the input sequence at all. Interestingly, the position of the balls under $(1, 0)$ seems to tend to the corners. When the temporal weights are unchanged and the visible weights scaled down the balls become indistinct and fuzzy although the motion of the balls is smooth. In the extreme case it becomes impossible to distinguish individual balls (See Fig. 5.2). The reconstructions under $(0.5, 0.5)$ show a mixture of the two effects.

Using the MOCAP dataset, the reconstructions failed to produce distinct qualitative shifts under various parameter shifts. The reconstructions produced under $(1, 0.5), (1, 0)$ and $(0.5, 1), (0.1)$ both show deficiencies in modeling the trajectory and the movement of the raw data. This suggests that the usefulness of this procedure depends highly on the nature of the dataset and the ability of an observer to interpret the results.

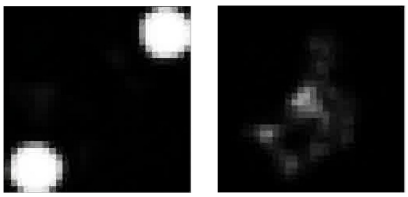

Figure 5.2: *A comparison of two sample frames from reconstructions of the bouncing balls under scaling factors (1,0) (on the left) and (0,1) (on the right). Under (1,0) one can see distinct balls. However, the balls stay mostly in the corners and exhibit very erratic movement. Under (0,1) distinct balls are no longer visible but the motion is smooth.*

# Chapter 6

# Conclusions

The idea of an artificial neural network has been around in some form or another since the 1940s. ANNs are appealing because of their simplicity and their biological foundation. Traditionally, the mathematical analysis of ANNs tends to focus on a particular model rather than discussing many different classes of models. This, combined with the search for models that produce desirable results, has led to a great deal of ambiguity as to what actually constitutes an ANN. By developing a common framework for the most relevant models we are able to capture concepts relevant to a wide variety of ANNs. From here we make precise several notions of universal approximation, allowing us to address the question of representational power in a way that is appropriate for different classes of networks. It should be noted that universal approximation of a stochastic process is in a sense a more general type of universal approximation than the one defined for globally stable probabilistic networks. This is because we may choose our stochastic process to be stationary, in which case the ability of a class of networks to approximate the process becomes equivalent to the ability of a class of networks to approximate a single distribution. Many popular neural networks were already known to satisfy the weaker definition of universal approximation, most notably, the restricted Boltzmann machine. This makes the RTRBM and TRBM good candidates for universal approximators of stochastic processes. Indeed, the intuition behind the proof of their universal approximation is quite straightforward, amounting to extending the universal RBM construction to universal (R)TRBM construction.

## 6.1 The TRBM vs the RTRBM

The proofs above have shown that generalized TRBMs and RTRBMs both satisfy the same universal approximation condition. In the proof of universal approximation for the RTRBM we take the weights large enough so that the real-valued hidden sequence becomes approximately binary. This suggests that the same proof could be adapted to the basic TRBM. However, the TRBM seems to have difficulty modeling distributions with long term time dependence. After an RTRBM was trained on videos of bouncing balls the machine was able to model the movement correctly and the hidden units contained chains of length two as described in the above proof [30]. On the other hand the TRBM did not have this structure and modeled the motion of balls as a random walk which is what one might expect for a machine that is unable to use velocity data by modeling two-step time dependencies. Given the likely equivalence in representational power of the two models, this discrepancy of results is best explained by the efficiency of the algorithm for the RTRBM in comparison to the TRBM.

## 6.2 Artificial Neural Networks and the Human Brain

Although inspired by biological neural networks, the study of ANNs quickly diverged from models concerned with biological realism. Despite this, universal approximation suggests that ANNs should at least, in theory, be able to do anything that could be accomplished with a more biologically accurate model. With this in mind, we can consider the potential use of ANNs in providing insight into the workings of the human brain. The similarities between biological neural networks and ANNs are for the most part superficial; however, there is a analogy that can be made between signaling pathways in the brain and distinct groups of parameters in ANNs. This allows for one to carry out experiments on perception with ANNs by the method of parameter variation. Realistic modelling of video sequences is still at this point computationally difficult. However, the universal approximation results show that this in not an intrinsic limitation of the current ANNs but rather the result of the learning algorithms being computationally time consuming or even in some cases potentially inaccurate. This may change in the future with the development of more efficient and reliable learning algorithms, or even simply by using more computational resources for the task of training ANNs on video data. The potential usefulness of experiments on ANNs via parameter variation is thus demonstrated through toy examples of bouncing balls and mocap data.

# Bibliography

[1] Y. Abu-Mostafa and JM. St. Jacques. Information capacity of the Hopfield model. *Transactions on Information Theory*, 31:461 – 464, 1985.

[2] S. Amari. Natural gradient works efficiently in learning. *Neural Computation*, 10:251–257, 1998.

[3] S. Amari, H. Park, and H. Ozeki. Geometrical singularities in the neuromanifold of multilayer perceptrons. *Advances in Neural Information Processing Systems*, 14:343 – 350, 2003.

[4] N. Ay, G. Montufar, and J. Rauh. Selection criteria for neuromanifolds of stochastic dynamics. *Advances in Cognitive Neurodynamics*, 3:147–154, 2013.

[5] SP. Franklin and M. Garzon. Neural computability. *Progress in Neural Networks*, 1:128,144, 1989.

[6] SP. Franklin and M. Garzon. Neural computability ii. *International Joint Conference on Neural Networks*, pages 631–637, 1990.

[7] Y. Freund and D. Haussler. Unsupervised learning of distributions of binary vectors using 2 layer networks. *Advances in Neural Information Processing Systems*, 4:912–919, 1991.

[8] G. Goodhill and M. Carreira-Perpinán. *Cortical Columns*. Macmillan, 1 edition, 2006.

[9] J. Hertz, A. Krogh, and R. Palmer. *Introduction to the Theory of Neural Computation*. Westview Press, new edition, 1991.

[10] G. Hinton. Training a product of experts by minimizing contrastive divergence. *Neural Computation*, 14:1771–1800, 2002.

[11] G. Hinton, D. Li, Y. Dont, and G.E. Dahl. Deep neural networks for acoustic modeling in speech recognition. *Signal Processing Magazine, IEEE*, 29:82–97, 2012.

[12] Geoffrey E. Hinton and Simon Osindero. A fast learning algorithm for deep belief nets. *Neural Computation*, 18:1527–1553, 2006.

[13] A. Hodgkin and A.F. Huxley. A quantitative description of membrane current and its application to conduction and excitation in nerve. *The Journal of Physiology*, 117:500–544, 1952.

[14] J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences of the USA*, 79:2554–2558, 1982.

[15] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4.

[16] O. Howes and S. Kapur. The dopamine hypothesis of schizophrenia: Version iii–the final common pathway. *Schizophrenia Bulletin*, 35, 2009.

[17] E.M. Izhikevich. *Dynamical Systems in Neuroscience*. MIT Press, 2010.

[18] A. Krizhevsky, I. Sutskever, and G. Hinton. Imagenet classification with deep convolutional neural networks. *Advances in Neural Information Processing Systems*, 25, 2012.

[19] G. Lebanon. Axiomatic geometry of conditional models. *IEEE Transactions on Information Theory*, 51:1283–1294, 2005.

[20] W. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115–133, 1943.

[21] M. Minsky and S. Papert. *Perceptrons*. MIT Press, 1969.

[22] G. Montufar and N. Ay. Refinements of universal approximation results for deep belief networks and restricted Boltzmann machines. *Neural Computation*, 23:1306–1319, 2011.

[23] G. Montufar, N. Ay, and K. Ghazi-Zahedi. Geometry and expressive power of conditional restricted Boltzmann machines. *Journal of Machine Learning(Preprint)*, 2014.

[24] N. Le Roux and Y. Bengio. Representational power of restricted Boltzmann machines and deep belief networks. *Neural Computation*, 20:1631–1649, 2008.

[25] N. Le Roux and Y. Bengio. Deep belief networks are compact universal approximators. *Neural Computation*, 22:2192 – 2207, 2010.

[26] D. Rumelhart, G. Hinton, and R. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986.

[27] M. Spies, G.M. Knudsen, R. Lanzenberger, and S. Kasper. The serotonin transporter in psychiatric disorders: Insights from PET imaging. *The Lancet Psychiatry*, 2:743–755, 2015.

[28] I. Sutskever and G. Hinton. Learning multilevel distributed representations for high-dimensional sequences. *Proceeding of the Eleventh International Conference on Artificial Intelligence and Statistics*, pages 544 – 551, 2006.

[29] I. Sutskever and G. Hinton. Deep, narrow sigmoid belief networks are universal approximators. *Neural Computation*, 20:2192 – 2207, 2010.

[30] I. Sutskever, G. Hinton, and G. Taylor. The recurrent temporal restricted Boltzmann machine. *Advances in Nueral Information Processing Systems*, 21:1601–1608, 2008.

[31] G. Taylor, G. Hinton, and S. Roweis. Modeling human motion using binary latent variables. *Advances in Neural Information Processing Systems*, 19:1345–1352, 2006.

[32] L. Younes. Synchronous Boltzmann machines can be universal approximators. *Applied Mathematics Letters*, 9:109–113, 1996.