Technische Universität München
(TUM)

Master Thesis

Deep Learning For Sequential Pattern Recognition

Author:
Pooyan Safari
pn.safari@gmail.com

Supervisor:
Martin Kleinsteuber
kleinsteuber@tum.de

A thesis submitted in fulfillment of the requirements
for the degree of Master of Science

in the

Geometric Optimization and Machine Learning
Faculty of Electrical Engineering and Information Technology

December 2013

Declaration of Authorship

I, Pooyan SAFARI, declare that this thesis titled, 'Deep Learning For Sequential Pattern Recognition' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: _____

Date: _____

"The truth is the whole. The whole, however, is merely the essential nature reaching its completeness through the process of its own development. Of the Absolute it must be said that it is essentially a result, that only at the end is it what it is in very truth; and just in that consists its nature, which is to be actual, subject, or self-becoming, self-development."

Hegel

TECHNISCHE UNIVERSITÄT MÜNCHEN (TUM)

Abstract

Faculty of Electrical Engineering and Information Technology
Department of Geometric Optimization and Machine Learning

Master of Science

Deep Learning For Sequential Pattern Recognition

by Pooyan Safari

In recent years, deep learning has opened a new research line in pattern recognition tasks. It has been hypothesized that this kind of learning would capture more abstract patterns concealed in data. It is motivated by the new findings both in biological aspects of the brain and hardware developments which have made the parallel processing possible. Deep learning methods come along with the conventional algorithms for optimization and training make them efficient for variety of applications in signal processing and pattern recognition. This thesis explores these novel techniques and their related algorithms. It addresses and compares different attributes of these methods, sketches in their possible advantages and disadvantages.

Acknowledgements

I would like to express my gratitude and special thanks to my supervisor, Professor Martin Kleinsteuber for his enthusiasm, patience and kindness. I have learned many aspects of carrying out research in the field of pattern recognition in the department of Geometric Optimization and Machine Learning in the faculty of Electrical Engineering and Information Technology of Technische Universität München (TUM), from his comments. I would also like to thank Clemens Hage for his advises and suggestions, and from our fruitful discussions. Without his encouraging and enlightening guidance, knowledge, persistent support this work would not have been successful. Many thanks are given to Professors Carlos Lopez Martinez, Juan-Antonio Fernandez Rubio, and Enrique Monte Moreno from Universitat Politècnica de Catalunya (Barcelona Tech.) who reviewed this thesis and all staffs who have helped me in one way or another and made the time of my master thesis pleasurable and memorable both at TUM and UPC.

Contents

List of Figures

List of Tables

Dedicated with a special feeling of gratitude to my loving parents, Catherine and Khosro whose words of encouragement and push for tenacity ring in my ears, and to my sister Paris, who is always special to me.

Chapter 1

Introduction

Pattern recognition has found its invaluable position in engineering. *Learning from data* and methods of analyzing data in the hope of extracting the concealed patterns plays an important role in different area of knowledge, from engineering to economics. One of these newly fashioned methods is called *deep learning*. It is a concept dated back to 1980s, however it was oppressed and ignored for a long time and gained attractions in 2000s. It is based on the *distributed representations* which introduced itself by connectionisms. These ideas in connection with the discoveries in the biological aspect of the brain make a rethinking about the deep architectures necessary.

Multiple level of representation is the principle idea behind the deep concept, it is inspired by this theory that the brain captures information in multiple layers, just as a hierarchy of filters each of which represents a portion of the information related to a specific phenomenon. These hierarchical models are possible to implement due to the advances occurred in the area of *high-speed general purpose graphical processing units (GPGPUs)*, and other hardware and software developments especially new algorithms for optimization and training.

1.1 Organization of the Thesis

Writing a thesis about the deep learning including different architectures, algorithms, mathematical modules is not an straight forward task. I tried to make the reader familiar with the sufficient background whenever needed. Thinking about deep learning without a background in *probabilistic graphical models*, artificial neural networks, probability theory and to some extent the optimization, is rather impossible.

Starting with a brief introduction to the pattern recognition with an intelligent system approach in chapter 2, I have tried to reveal the key points of deep learning. It has been attempted to cover both conceptual and technical aspects as far as it was possible. This is done by a taxonomy of different elements involving in a pattern recognition task, namely *representation*, *inference*, *learning*. This is followed by a primary introduction to the *probabilistic graphical models* which is the basis for the *Boltzmann machines*.

In chapter 3 the basics of neural networks is introduced followed by the new generation of neural networks which is called *energy-based* models. One may find details about the *Boltzmann machines* and the training algorithm which is quite common nowadays.

Chapter 4, the deep learning concept and philosophy is introduced. In later sections of this chapter, one could find the basic deep architectures which are mainly used for non-sequential patterns. A complete discussion and overview about the deep architectures concerning sequential patterns is taken place in chapter 5.

Multiple sources and references (more than 200 references) have been used, this provides readers with different possible approaches which makes the comparisons more sensible. For instance, in the case of neural networks in chapter 3, both biological and mathematical model for neurons are discussed, or in the case of deep architectures, both philosophical-historical background and practical point of view are considered.

For the notation, small bold characters are used to show the vectors and capital bold characters represent the matrices. Other specific notations will be explained whenever used throughout the thesis.

Chapter 2

Pattern Recognition

During the daily routine we deal with different Patterns. When we identify our friends from their voice or understand a telephone conversation, or detect the genre of the music broadcast by a radio station, we recognize the complex patterns. Recognizing the Pattern or Pattern Recognition is the process of taking in raw data and taking an action based on the *category* of the pattern [Duda et al., 2012] if there is any.

In this Chapter, the basic concepts of pattern recognition is introduced, focused mainly on a conceptual understanding of the whole procedure.

2.1 Elements of Pattern Recognition

In pattern recognition we are seeking to find the ways and methods to design machines that better recognize patterns. One natural way of doing so, is to understand the process done by the nature itself so that it extracts the patterns from the sensory data. If we assume that the task done by the nature is perfect enough to recognize the patterns then one logical solution would be tending to the point to do exactly the same action. In order to do so we have to understand what and how nature acts and then try to interpret and model it into a language suitable for the machines. There are three critical characteristics for any intelligent system, namely *representation*, *inference*, and *learning* [Koller and Friedman, 2009]. Pattern recognition includes elements which are briefly introduced in the following subsections. Note that these are not isolated elements with distinct borders, or multiple steps followed one by another, but rather they are the characteristics that one may need to call a system intelligent and sometimes it is really hard to make a distinction between these terms during a process. One may find a mixture of them in different steps of a pattern recognition task. I would like to emphasize that,

although these terms (representation, inference, and learning) have been used by other authors quite frequently, here a slightly different tasks (sometimes a broader meaning, especially in the case of representation) are assigned to them.

2.1.1 Representation

Our machines, which mainly consist of digital processors, work with numbers or better to say arrays of numbers. For instance, a video camera which records the images, will output an array of pixels each with a particular gray level or color. You might get a square array of 512 by 512 such pixels, and each pixel value would, on a gray scale, perhaps, be represented by a number between 0 (black) and 255 (white). If the image is in color, there will be three such numbers for each of the pixels, say the intensity of red, blue and green at the pixel location. The numbers may change from system to system and from country to country, but you can expect to find, in each case, that the image may be described by an array of *real* numbers, or in mathematical terminology, a vector in $\Re^n$ for some positive integer n. The number n, the length of the vector, can therefore be of the order of a million. For instance, to describe the image of the screen, which has 1024 by 1280 pixels and a lot of possible colors, I would need $3,932,160$ numbers, $\Re^{3,932,160}$ [Alder, 1997].

So we have to interpret (define, identify, invent or discover), say *represent*, all the *things*, say *features*, that a specific phenomenon comprises, albeit describing a phenomenon in terms of quantized, discrete *things* is of philosophical interest, whether it is accurate enough or just a misunderstanding of the world. This *interpretation* might be done manually, by the human with specific knowledge about the certain phenomenon, automatically by the machine itself (*feature learning*), or a combination of the both, a *tandem* approach.

One may think of the *representation* as a mapping from the *object* (phenomenon) to the points in mathematical space, also we can call it *coding* of the object [Alder, 1997]. For instance assume a microphone monitoring sound levels, there are many ways of coding the signal. It can be simply a matter of a voltage changing in time, that is, $n = 1$. Or we can take a Fourier Transform and obtain a simulated filter bank, or we can put the signal through a set of hardware filters. In these cases n may be, typically, anywhere between 12 and 256 [Alder, 1997].

After defining the *features* you have to compute these *features*, *feature extraction*, this is what usually mixed up with *feature learning*. Actually there is a close relation between them and sometimes it is quite impossible to distinguish one from another, either of the

two is used for representing the input phenomenon. Sometimes they are done at the same time. However, the concept is different.

We might sometimes do some pre-processing stages in order to make the pattern recognition task easier, faster or less expensive. Feature extraction might be thought of as one of these stages which is also a kind of *dimension reduction*. As another example assume real-time face detection in a high resolution video stream where computers must handle huge numbers of pixels per second, and presenting these directly to a complex pattern recognition algorithm may be computationally infeasible (expensive). Instead, the aim is to find useful features that are fast to compute, and yet that also preserve useful discriminatory information enabling faces to be distinguished from non-faces. These features are then used as the inputs to the pattern recognition algorithm. For instance the average value of the image intensity over a rectangular subregion can be evaluated extremely efficient [Viola and Jones, 2004], and a set of such features can prove very effective in fast face detection. Because the number of such features is smaller than the number of pixels, this kind of pre-processing represents a form of dimensionality reduction. Care must be taken during pre-processing because often information is discarded, and if this information is important to the solution of the problem then the overall accuracy of the system can suffer [Bishop, 2006]. In order to have a more abstract representation, you should make use of *deeper features*.

Another issue for *representation* is the concept of *model* and the task of *model selection*. The term *model* has a broad range of functionality. There are two different classes of models, *deterministic* where no randomness is considered and usually designed by the human himself like the second Newton's law of motion $f = ma$, and *statistical* models where uncertainty and randomness are the principles. Despite the fact that in many cases we have no idea whether a phenomenon (e.g., universe) is deterministic itself or our lack of knowledge results in an uncertain interpretation of it, the important thing is, with our current knowledge there is no chance for a certain understanding of the phenomenon and drawing deterministic conclusions out of it. Some models describe the interaction between different *things* involved in a system, such as *hidden Markov model*, or describe a specific variable through the system, such as Gaussian distribution, or reflect the input-output relation of the system. For example, a model for medical diagnosis might represent our knowledge about different diseases and how they relate to a variety of symptoms and test results. A reasoning algorithm can take this model, as well as observations relating to a particular patient, and answer questions relating to the patient's diagnosis [Koller and Friedman, 2009].

Assume we train a model $\mathcal{M}^1$ which is a polynomial with degree 2 and we get a training error $\mathcal{E}^1$, now we train again the model $\mathcal{M}^2$, on the same training data, which is a

polynomial of degree nine and we get $\mathcal{E}^2$ [Koller and Friedman, 2009]. How do we decide which model is the best, is the problem of *model selection*. In the previous example, the order of the polynomial controls the number of free parameters in the model and thereby governs the model complexity, however we might use more complex models such as *mixture distributions*, *neural networks*, *support vector machines*, or even *graphical models* such as *Bayesian networks* and *Markov random fields*.

This thesis deals with statistical models. Statistical models are divided into two subcategories, *parametric* and *non-parametric* models. A *statistical model* $\mathfrak{F}$ is a set of distributions (or densities or regression functions). A parametric model is a set $\mathfrak{F}$ that can be parameterized by a finite number of parameters, such as the Gaussian model for density. On the contrary, a non-parametric model is a set $\mathfrak{F}$ that cannot be parameterized by a finite number of parameters [Wassermann, 2003]. The term non-parametric does not imply that such models completely lack parameters but that the number and nature of the parameters are flexible and not fixed in advance. For instance, a *histogram* is a simple non-parametric estimate of a probability distribution.

2.1.2 Inference

As it is mentioned before we are dealing with statistical pattern recognition, so we would cope with *statistical inference*. In statistics, statistical inference is the process (one can think of it as the methodology) of drawing conclusions about a population on the basis of measurements or observations made on a sample of units from the population [Everitt and Skrondal, 2010]. This is the most important question of the inference process, *given the outcomes, what can we say about the process that generated the data?*. *Prediction*, *classification*, *clustering*, and *estimation* are all special cases of statistical inference. Data analysis, machine learning, and data mining are various names given to the practice of statistical inference, depending on the context [Wassermann, 2003]. Any statistical inference requires some assumptions, say models, section 2.1.1. There are many approaches, say *schools* or *paradigms*, to statistical inference, such as *frequentist inference*, *Bayesian inference*, *Fiducial inference*, *structural inference*, *information and computational complexity*. These paradigms are not mutually exclusive, and methods which work well under one paradigm often have attractive interpretations under another. The two main paradigms, mostly used nowadays, are *frequentist* and *Bayesian inference*. Among the parametric models we choose the *maximum likelihood* method of inference and Bayesian inference one from the frequentist approach and the other Bayesian approach, however we can use non-parametric methods such as *Bootstrap* which is also a frequentist approach [Wassermann, 2003].

So far we have seen that the inference itself might be classified as frequentist versus Bayesian methods, methods related to parametric models versus non-parametric models (in short parametric versus non-parametric), however there is anther classification which is *generative* versus *discriminative*. For instance, the maximum likelihood method is a parametric, frequentist, generative method of inference.

In the generative case we have to first determine the *likelihood*[1] and then using the Bayes' theorem compute the *posterior density*[2]. On the other hand it is also possible to determine the posterior densities directly without any additional transformation, this is the so called discriminative method.

2.1.3 Learning

Although the ability to retain, process and project prior experience onto future situations is indispensable, the human mind also possesses the ability to override experience and adapt to changing circumstances. Memories of individual events are not very useful in themselves, but, according to the received view, they form the raw material for further learning. By extracting the commonalities across a set of related episodic memories, we can identify the underlying regularity, a process variously referred to as *abstraction*, *generalization* or *induction* [Ohlsson, 2011]. The problem of learning can be considered as the problem of change. When you learn, you change the way that information is processed by the system [O'Reilly and Munakata, 2000].

The *learning* or *training* element, refers to methods that we use in order to adjust and modify the parameters of the system or model to better fit to training data, which is usually equivalent to an iterative optimization process done by means of some algorithms.

Learning can be divided in three broad groups of algorithms, namely, *supervised learning*, *unsupervised learning*, and *reinforcement learning*.

In supervised learning (sometimes called *associative learning*) we are trying to predict an output when given an input vector. It comes in two different classes, *regression* and *classification*. In regression the target output is a real number or a whole vector of real numbers, such as a price of a stock during six months, or the temperature of a room. The aim here is to get as close as we can to the correct real number. In classification, the target output is a class label, the simplest case is a choice between 1 and 0, between positive and negative cases. However, we may have multiple alternative labels as when

[1] If x and y are two random variables and you assume that y has been observed (we mean it is known to occur or to have occurred) then $p(x|y)$ is the posterior probability and $p(y|x)$ is called likelihood. Remember from probability theory whenever we are talking about $p(x|y)$, we are thinking of y as a parameter not a random variable, albeit y is a random variable in essence.

[2] See footnote 1

we are classifying handwritten digits. Supervised learning works by initially selecting a model-class, that is a whole set of models that we are prepared to consider as candidates. A model-class, f, is a way of using some numerical parameters, $\boldsymbol{W}$, to map each input vector $\boldsymbol{x}$, into a predicted output y, and then adjust these numerical parameters to make the mapping fit the supervised training data. What is meant by fit is minimizing the discrepancy between the target output on each training case and the actual output produced by a machine learning system. A natural measure for the discrepancy, when we are dealing with real valued data as outputs, is the *squared difference* $(y-t)^2$, where t is the target output. However there are other choices for discrepancy measure, which depend on the characteristics of the applications.

In unsupervised learning (also sometimes referred to as *density estimation* problems or *self-organization*) we are trying to discover a good internal representation of the input. One major aim is to create an internal representation of the input that is useful for subsequent supervised or reinforcement learning. The reason we might want to do that in two stages, is we don't want to use, for example, the payoffs from reinforcement learning in order to set the parameters for our visual system. So we can compute the distance to a surface by using the disparity between images we get in our two eyes, however we don't want to learn to do that computation of distance by repeatedly stubbing our toe and adjusting the parameters in our visual system every time we stub our toe. That would involve stubbing our toe a very large number of times and there are much better ways to learn to fuse two images based purely on the information in the inputs. Other goals for unsupervised learning are to provide compact, low dimensional representations of the input, so high-dimensional inputs like images, typically live on or near a low-dimensional manifold or several such manifolds, what that means is even if you have million pixels, there are not really a million degrees of freedom in what may happen, there may only be a few hundred degrees of freedom in what can happen, so what we want to do is to move from a million pixels to a representation of those few hundred degrees of freedom which will be according to saying where we are on a manifold, also we need to know which manifold we are on. A very limited form of this, is *principle component analysis* which is linear. It assumes that there is one manifold, and the manifold is a plane in the high dimensional space. Another definition of unsupervised is to provide economical representation for the input in terms of learned features. If for example, we can represent the input in terms of binary features, that's typically economical because then it takes only one bit to say the state of a binary feature, alternatively we could use a large number of real-valued features but insist that for each input almost all of those features are exactly zero. In that case for each input we only need to represent a few real numbers and that is economical. As mentioned before, another goal of unsupervised learning is to find clusters in the input, and clustering could be viewed as a very *sparse code*, that

is we have one feature per cluster, and we insist that all the features except one are zero and that one feature has a value of one. So clustering is really just an extreme case of finding sparse features.

A practical example of the application of unsupervised learning involves the interpretation of X-ray images (mammograms) used for breast cancer screening [Tarassenko et al., 1995]. The goal is to model the unconditional distribution of data described by some vector $\boldsymbol{x}$. In this case the training vectors $\boldsymbol{x}$ form a sample taken from normal (non-cancerous) images, and a network model is used to build a representation of the density $p(\boldsymbol{x})$. When a new input vector $\boldsymbol{x'}$ is presented to the system, a high value for $p(\boldsymbol{x'})$ indicates a normal image while a low value indicates a novel input which might be characteristic of an abnormality. This is used to label regions of images which are unusual, for further examination by an experienced clinician [Jordan and Bishop, 1996].

In reinforcement learning we are trying to select actions or sequences of actions to maximize the rewards, and the rewards may only occur occasionally. In other words, the output is an action or sequence of actions and the only supervisory signal is an occasional scalar reward. The goal in selecting each action, given a particular state, is to maximize the expected sum of the future rewards, and we typically use a discount factor for delayed rewards so that we do not have to look too far into the future. Reinforcement learning is a difficult task since the rewards are typically delayed so it is hard to know where we went wrong or right, and also a scalar reward does not supply much information on which to base the changes in parameters. Consequently, we cannot learn millions of parameters using reinforcement learning whereas in supervised learning and unsupervised learning we can. Typically the number of parameters is in order of dozens or 1000 parameters.

2.2 Sequential Pattern Recognition

In many applications we deal with *identically independent distribution* which is usually called in short *i.i.d.*, this is based on the assumption that in a specific phenomenon, random variables are independent one from another, but follow the same distribution. In this case we can conclude from the *product rule*[1] of probability and rewrite the *joint density* and *likelihood* as the product over all data points of the probability distribution at each data point. However, in many other applications, the assumption of statistically independence is not enough and results in poor conclusions. One of such classes of data is the *sequential* data. The term sequential has a broad meaning, it is not only referred

[1]Probability consists of two rules, sum rule $p(x) = \sum_y p(x, y)$, product rule $p(x, y) = p(x)p(y|x)$. Now if x and y are statistically independent we can write $p(y|x) = p(y)$ which concludes with $p(x, y) = p(x)p(y)$.

to the data which are temporal but also to any kind of data which are static in essence while comprises and ordered list of events [Roddick and Spiliopoulou, 2002]. These situations often arises in *time series*[1], or the *temporal sequences* such as the sequence of nucleotide base pairs along a stand of DNA or the sequence of characters in an English sentence [Bishop, 2006].

As mentioned in 2.1.1, we have to represent the *things* which are sufficient to interpret a phenomenon, say pattern here. One of the main *things* is sequence order here, in other words we have to represent the dependencies between different elements of a sequence to have a better understanding of the pattern. Although, it is up to the engineer or designer of the system to simply ignore the sequential aspects (characteristics) of the data and treat data as a series of actions which are independent one from another and of course this makes the analysis more convenient. The information which are concealed in this interaction (between data points) would be lost, which might be critical for the final analysis of the system, here pattern. You can see the graphical interpretation with i.i.d. assumption in Figure 2.1. For instance in the case of rainfall measurement if we ignore the sequential correlations between consecutive days and treat the measurements as i.i.d. data points, given a specific day we can not predict the rainfall for the successive day or days [Bishop, 2006]. So in order to express such effects in a probabilistic models, we need to relax the i.i.d. assumption.

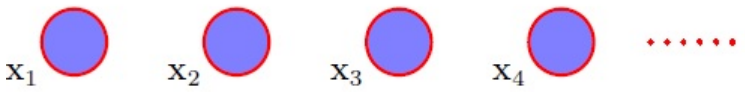


FIGURE 2.1: The simplest approach to modeling a sequence of observations is to treat them as independent, corresponding to graph without links [Bishop, 2006].

There are different methods, say models, in order to show the sequential characteristics of the patterns, such as *Dynamic Bayesian Networks*, *Linear Dynamical Systems*, *Sliding Window Method*, *Recurrent Sliding Windows*, *conditional random fields*, *mixture distributions*, *Hidden Markov Models*, *Discriminative methods*, *maximum entropy Markov models*, *graph transformer networks*, *input-output Markov models* and so on. The *Markov model* is amongst the invaluable models to represent the sequential patterns. It is a member of *probabilistic graphical models*' family. Therefore we introduce

[1] A time series is a sequence (series) of data points, measured typically at successive points in time spaced at uniform time intervals. For instance the daily closing value of Dow Jones or NASDAQ index, or the rainfall measurement of a specific region during a sequence of days, or the acoustic features at successive time frames used for speech recognition. The time series method of analysis might be divided into two subcategories namely, the *frequency-domain* methods and *time-domain* methods. The former includes spectral analysis and wavelet analysis, the latter includes auto-correlation and cross-correlation analysis. It is different from spatial data analysis where the observations typically relate to geographical locations, e.g., accounting for house prices by the location as well as the intrinsic characteristics of the houses.

the *probabilistic graphical models* briefly which guides us to deeper understanding of *Markov model* and its successor *hidden Markov model* and would be useful in other chapters especially in section 3.3 for better understanding *the new generation of neural networks*.

2.2.1 Probabilistic Graphical Models

Probability theory can be expressed in terms of two simple equations see footnote 1 of page 9 corresponding to the sum rule and the product rule. So we could take advantage of these two rules so as to reformulate and make the complex probabilistic models more convenient to cope with, and we have a good opportunity since they are only simple algebraic formulations. Another idea, which is quite often in science, is to transform a problem from one domain into another, and exploit the properties and characteristics of the destination domain in order to better understand the original problem and make the computations and solutions more convenient. This is what happens when we transit from time domain to the frequency domain in signal processing. Here we use another transition from probabilistic models to the *graph* representation so as to benefit from the *graph theory* properties and algorithms or methods involved in this area. This new domain is called the *probabilistic graphical models*.

There are two kinds of *probabilistic graphical models*, one is *directed graphical models* (*acyclic*) also known as *Bayesian network*, and the other is *undirected graphical model* or *Markov random field* or *Markov network*. They are useful in different structures, for instance while the directed graphs are valuable for expressing the causal sequence of random variables, the undirected one are better at interpreting the soft constraints between random variables [Bishop, 2006].

For the purposes of solving inference problems, it is often convenient to convert these diagrams (graphs) into a different representation called a *factor graph*. Actually the main intention is to use graph representation in order to take advantage of graph theory and try to change complex probabilistic inferences considerably easier. In the probabilistic graphical representation, illustrated in Figure 2.2, the nodes correspond to the random variables in our domain, and the edges correspond to direct probabilistic interactions between them [Koller and Friedman, 2009].

Regarding the Figure 2.2, a Bayesian network, the joint probability distribution $p(a, b, c)$ over three variables a, b, and c might be written in the form:

$$p(a, b, c) = p(a)p(b|a)p(c|a, b) \tag{2.1}$$

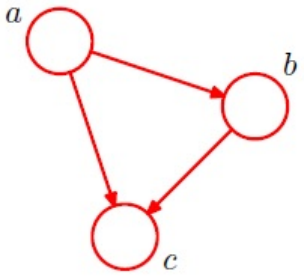


FIGURE 2.2: A directed graphical model representing the joint probability distribution over three variables a, b, and c, corresponding to the decomposition on the right-hand side of equation 2.1 [Bishop, 2006].

Figure 2.2 is called a fully connected graph, since there is a link between every pair of nodes. The absence and existence of an edge in a graph are both informative about the probability distributions [Bishop, 2006]. Now assume a more complicated graph in Figure 2.3, where the joint distribution $p(x_1, \ldots, x_7)$ is given by

$$p(x_1)p(x_2)p(x_3)p(x_4|x_1, x_2, x_3)p(x_5|x_1, x_3)p(x_6|x_4)p(x_7|x_4, x_5) \tag{2.2}$$

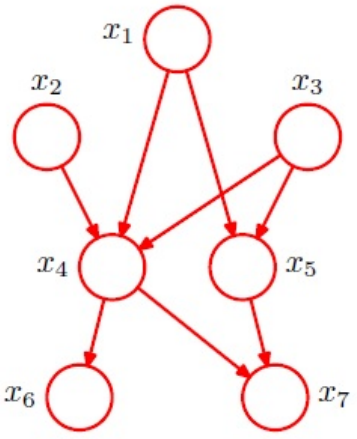


FIGURE 2.3: Example of a directed acyclic graph describing the joint distribution over variables $x_1, \ldots, x_7$. The corresponding decomposition of the joint distribution is given by 2.2 [Bishop, 2006].

Equation 2.2 is called a *factorization* of the joint distribution $p(x_1, \ldots, x_7)$. An important concept for probability distributions over multiple variables is that of *conditional independence*. a and b are conditionally independent given c, if $p(a, b|c) = p(a|c)p(b|c)$, see Figure 2.4 for graphical representation.

Writing the joint probability of each graph of Figure 2.4 and using the Bayes' theorem one would find that a and b in the graphs on the left and in the middle are conditionally

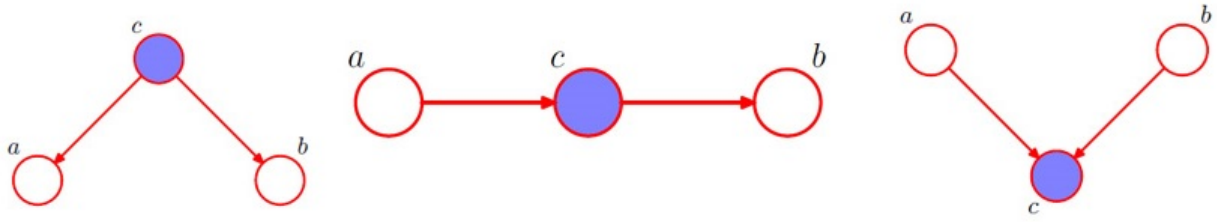


FIGURE 2.4: Graph representation of conditional dependencies. Left: this a tail-tail according to node c, a and b are conditionally independent given c. Middle: c is a tail-head node, a and b are conditionally independent given c. Right: c is a head-head node, a and b are not conditionally independent [Bishop, 2006].

independent given c, while on the right they are not conditionally independent given c, although a and b are still marginally independent from each other.

In order to use the tools and properties from graph theory, we have to introduce some basic definitions. The node c in the Figure 2.4 is tail-tail, head-tail, and head-head respectively (from left to right). Assume the graph in Figure 2.5, nodes e and c are both *descendants* of node a, b and e are also *descendants* of node f. Starting from a node, assume e, and going in the direction of the arrows, all the nodes that you pass are the *descendants* of the starting node, here only c. According to this definition, there is no descendant for node b in Figure 2.5. Now assume G is a directed acyclic graph (DAG), and let A, B, and C are disjoint subsets of vertices (the union of these subsets may not include all the vertices in G)of G. A path in a graph is a sequence of vertices for which there is an edge in the graph for each pair of successive vertices in the sequence, it is not necessarily directed, for instance in Figure 2.5, the sequences $\{a, e, c\}$, and $\{a, e, f, b\}$ are both paths. A path between two vertices is *blocked* with respect to C, if it passes through a vertex ν such that either one of the properties holds [Bishop, 2006]:

1. the arrows at vertex ν are either head-tail or tail-tail, and the vertex is in the set C.

2. the arrows at vertex ν are head-head, neither the vertex ν, nor any of its *descendants*, is in the set C.

For instance, in the graph (a) of Figure 2.5 the path between nodes a and b is not blocked by node f, because node f is a tail-tail node while it is not in the observed set (the conditioned set). The node e does not block the path, since it is a head-head node while its descendant (node c) is in the observed set. On the contrary, in the graph (b), the path is blocked by f, it is tail-tail node and is observed. It is also blocked by node e because e is a head-head node and none of its descendants is in the observed set.

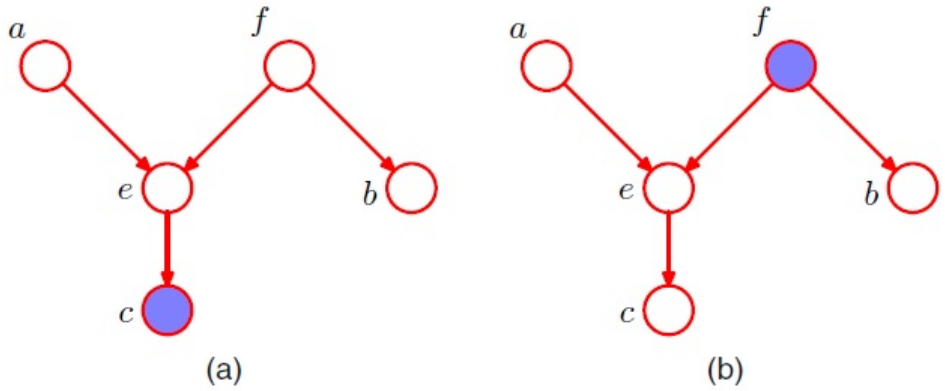


FIGURE 2.5: Illustration of the concept of d-separation [Bishop, 2006].

The whole point of graphical models is to express the conditional independence properties of a probability distribution and the d-separation criterion gives you a way to get rid of those conditional independence properties from a graphical model for a probability distribution. Now back to our subset of nodes A, B, and C, we would say A and B are *d-separated* by C if all paths from any vertex (node) of set A to any vertex of set B are blocked with respect to C. The important consequence of this definition according to the d-separation theorem is, if A and B are d-separated by C, then A and B are conditionally independent given C. Note that the converse to the theorem cannot be always true, it is possible that A and B are conditionally independent while they are not d-separated given C.

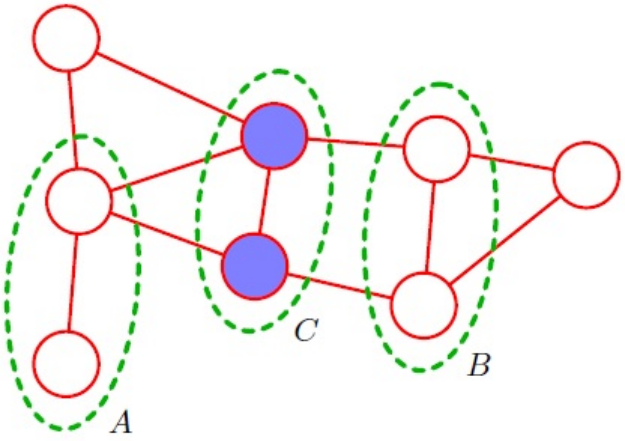


FIGURE 2.6: An example of an undirected graph in which every path from any node in set A to any node in set B passes through at least one node in set C. Consequently the conditional independence property holds for any probability distribution described by this graph [Bishop, 2006].

Another type of probabilistic graphical models, as mentioned before, is the *Markov random field*, which is called also *Markov network* or *undirected graphical model*. In this

kind of representation, there is a set of nodes each of which corresponds to a random variable or group of variables, as well as a set of edges each of which connects a pair of nodes. Here the edges are undirected, that is they do not carry arrows, or sometimes we can say they have arrows in both directions. As we did before for the case of directed graphs, we shall discuss about conditional independence properties and its definitions in this kind of graphs [Bishop, 2006].

Assume three sets of nodes such that in Figure 2.6, there are ways that we can test whether conditional independence property holds given C, the first way is to consider all possible paths that connect nodes in set A to nodes in set B. If all such paths pass through one or more nodes in set C, then all such paths are *blocked* and so the conditional independence property holds. However, if there is at least one such path that is not *blocked*, then the property does not necessarily hold, or in other words it means that there is at least some distributions corresponding to the graph that do not satisfy this conditional independence property. This is in some way similar to what we have discussed for the d-separation criterion and only differs in the case of *explaining away*[1] phenomenon, since there is no such effect. There is also another way to test whether the conditional independence property holds, which is to remove all the nodes and their corresponding edges in the set C, then try to find a path that connects *any* node in A to *any* node in B. If you couldn't find any then the conditional independence property must hold [Bishop, 2006].

Now we need a kind of factorization rule, as mentioned before for the directed graphs, using the conditional independence properties of the graph. Here we have to introduce another graphical concept called *clique*. A clique is a subset of the nodes which together constitute a fully connected graph, in other words all the nodes and/or subsets of notes link one another. Furthermore, among the cliques, one which is not possible to comprise any other node of the graph without it ceasing to be a clique anymore, is called a *maximal clique*. Figure 2.7 is a graph over four variables and the concepts of clique and maximal clique are illustrated. This graph has five cliques of two nodes given by $\{x_1, x_2\}, \{x_2, x_3\}, \{x_3, x_4\}, \{x_4, x_2\}, \{x_1, x_3\}$, as well as two maximal cliques given by $\{x_1, x_2, x_3\}, \{x_2, x_3, x_4\}$. The set $\{x_1, x_2, x_3, x_4\}$ is not a clique because of the missing link from x_1 to x_4 [Bishop, 2006].

Let us denote a clique by C, and the set of nodes (random variables) in that clique by $\boldsymbol{x}_C$. Then the joint distribution is written as a product of *potential functions* $\psi_C(\boldsymbol{x}_C)$ over the maximal cliques of the graph

[1]The *explaining away* phenomenon says that if you have competing possible reasons for *explaining* some event, and the chances of one of those reasons increases, the chances of the others must decline since they are being *explained away* by the first *explanation*. In other words those reasons (causes) might be conditionally dependent even though they are marginally independent.

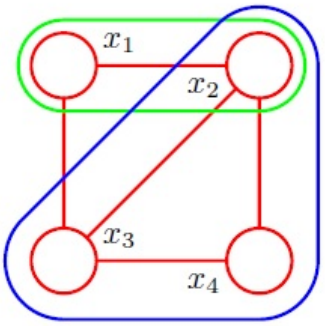


FIGURE 2.7: A four-node undirected graph showing a clique (outlined in green) and a maximal clique (outlined in blue) [Bishop, 2006].

$$p(\boldsymbol{x}) = \frac{1}{Z} \prod_{C} \psi_C(\boldsymbol{x}_C) \tag{2.3}$$

One may think of it as a kind of factorization by means of potential functions. We shall consider the constraint $\psi_C(\boldsymbol{x}_C) \geqslant 0$ for the potential functions in order to ensure that $p(\boldsymbol{x}) \geqslant 0$ is satisfied. So far we have not imposed any constraint on the choice of potential functions, except the non-negativeness of the potentials. In directed graphs each factor represents the conditionality of the corresponding variables, however it is not explicit in the case of undirected graphs. This is the difference that results in the definition of cliques and maximal cliques so as to introduce conditionality in terms of these new concepts instead of assigning a distinct probability distribution to each edge of the graph.

However, in special cases, for instance where the undirected graph is constructed by starting with a directed graph, the potential functions may indeed have such an interpretation.

One consequence of the generality of the potential functions $\psi_C(\boldsymbol{x}_C)$ is that their product will in general not be correctly normalized. We therefore have to introduce an explicit normalization factor given by 2.4. In the case of directed graphs, we did not need this normalization, since we had given a probability interpretation to each link of the graph so it had been normalized. Here the quantity Z, sometimes called the *partition function*, is a normalization constant and is given by

$$Z = \sum_{x} \prod_{C} \psi_C(\boldsymbol{x}_C) \tag{2.4}$$

which ensures that the distribution $p(\boldsymbol{x})$ given by 2.3 is correctly normalized. For example in Figure 2.7, one may write the joint probability as

$$p(A = a_1, B = b_1, C = c_1, D = d_1) = \frac{1}{Z} [\psi_{BCD}(B = b_1, C = c_1, D = d_1)\psi_{AB}(A = a_1, B = b_1) \psi_{DA}(D = d_1, A = a_1)] \quad (2.5)$$

while the partition factor Z for this example is given as follows

$$Z = \psi_{BCD}(B = b_1, C = c_1, D = d_1)\psi_{AB}(A = a_1, B = b_1)\psi_{DA}(D = d_1, A = a_1) + \cdots + \psi_{BCD}(B = b_N, C = c_N, D = d_N)\psi_{AB}(A = a_N, B = b_N)\psi_{DA}(D = d_N, A = a_N) \quad (2.6)$$

we can even expand the $\psi_{BCD}(B, C, D)$ and give it as a multiplication of more explicit factors $\psi_{BC}(B, C)\psi_{CD}(C, D)\psi_{DB}(D, B)$.

We now seek a relation between the factorization and conditional dependencies. To do so, we constrain ourselves to the potential functions $\psi_C(\boldsymbol{x}_C)$ that are strictly positive (i.e., never zero or negative for any choice of $\boldsymbol{x}_C$).

Since we are restricted to potential functions which are strictly positive it is convenient to express them as exponentials, so that

$$\psi_C(\boldsymbol{x}_C) = exp\{-E(\boldsymbol{x}_C)\} \quad (2.7)$$

where $E(\boldsymbol{x}_C)$ is called an *energy function*, and the exponential representation is called the *Boltzmann distribution.* The joint distribution is defined as the product of potentials, and so the total energy is obtained by adding the energies of each of the maximal cliques.

In contrast to the factors in the joint distribution for a directed graph, the potentials in an undirected graph do not have a specific probabilistic interpretation. Although this gives greater flexibility in choosing the potential functions, because there is no normalization constraint, it does raise the question of how to motivate a choice of potential function for a particular application. This can be done by viewing the potential function as expressing which configurations of the local variables are preferred to others. Global configurations that have a relatively high probability are those that find a good balance in satisfying the (possibly conflicting) influences of the clique potentials.

For a more thorough discussion about the probabilistic graphical models I can refer you to the books such as [Koller and Friedman, 2009; Whittaker, 1990; Jordan, 1999; Lauritzen, 1996; Cowell et al., 1999]

2.2.2 Markov Models

The probabilistic graphical models have been quite general so far, we have not assumed anything about the sequence of data. They might be used equally for sequential and non-sequential patterns. However, we now turn to a more specific interpretation (representation) which is suitable for sequential data, the *hidden Markov model*. We start with *Markov chain* which is the logical predecessor of *hidden Markov model*.

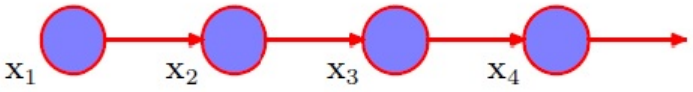


FIGURE 2.8: A first-order Markov chain of observations $\{x_n\}$ in which the distribution $p(x_n|x_{n-1})$ of a particular observation x_n is conditioned on the value of the previous observation x_{n-1} [Bishop, 2006].

As it is clear from the Figure 2.8, the joint distribution for a sequence of N observations under this model is given by

$$p(x_1, \ldots, x_N) = p(x_1) \prod_{n=2}^{N} p(x_n|x_{n-1}) \tag{2.8}$$

We can simply consider an M^{th} order Markov chain which means that the conditional distribution for a particular variable depends on the previous M variables. A second-order ($M = 2$) Markov chain is illustrated in Figure 2.9 and the joint distribution is given by

$$p(x_1, \ldots, x_N) = p(x_1)p(x_2|x_1) \ldots p(x_M|x_{M-1}, \ldots, x_1) \prod_{n=M+1}^{N} p(x_n|x_{n-1}, \ldots, x_{n-M}) \tag{2.9}$$

Using d-separation or by direct evaluation, we see that the conditional distribution of $\boldsymbol{x}_n$ given $\boldsymbol{x}_{n-1}$ and $\boldsymbol{x}_{n-2}$ is independent of all observations $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{n-3}$. Each observation is now influenced by only two previous observations.

Suppose we wish to build a model for sequences that is not limited by the Markov assumption to any order and yet that can be specified using a limited number of free

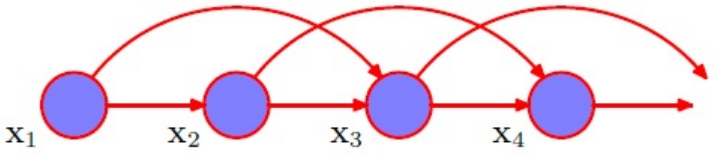


FIGURE 2.9: A second-order Markov chain, in which the conditional distribution of a particular observation x_n depends on the values of the two previous observations x_{n-1} and x_{n-2} [Bishop, 2006].

parameters. We can achieve this by introducing additional latent variables to permit a rich class of models to be constructed out of simple components, as what is done with mixture distributions. For each observation $\boldsymbol{x}_n$, we introduce a corresponding latent variable $\boldsymbol{z}_n$ (which may be of different type or dimensionality to the observed variable). We now assume that it is the latent variables that form a Markov chain, giving rise to the graphical structure known as a state space model, which is shown in Figure 2.10. It satisfies the key conditional independence property that $\boldsymbol{z}_{n-1}$ and $\boldsymbol{z}_{n+1}$ are independent given $\boldsymbol{z}_n$. The joint distribution for this model is given by

$$p(x_1, \ldots, x_N, z_1, \ldots, z_N) = p(z_1) \left[\prod_{n=2}^{N} p(z_n|z_{n-1}) \right] \prod_{n=1}^{N} p(x_n|z_n) \tag{2.10}$$

where $p(z_1)$ and $p(z_n|z_{n-1})$ are *transition* probabilities and $p(x_n|z_n)$ are *emission* probabilities. Using the d-separation criterion, we see that there is always a path connecting any two observed variables x_n and x_m via the latent variables, and that this path is never blocked. Thus the predictive distribution $p(x_{n+1}|x_1, \ldots, x_n)$ for observation x_{n+1} given all previous observations does not exhibit any conditional independence properties, and so our predictions for x_{n+1} depends on all previous observations. The observed variables, however, do not satisfy the Markov property at any order.

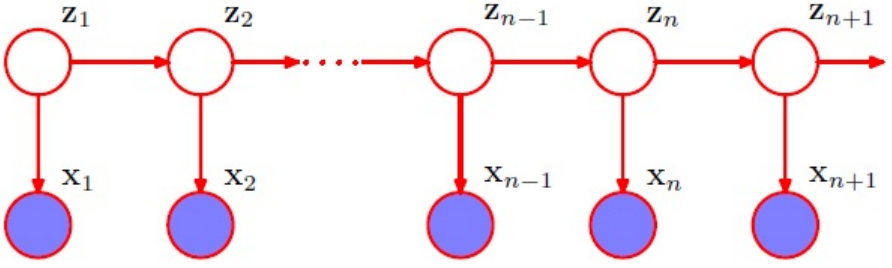


FIGURE 2.10: We can represent sequential data using a Markov chain of latent variables, with each observation conditioned on the state of the corresponding latent variable. This important graphical structure forms the foundation both for the *hidden Markov model* and for *linear dynamical systems* [Bishop, 2006].

There are two important models for sequential data that are described by this graph. If the latent variables are discrete, then we obtain the *hidden Markov model*, or HMM. Note

that the observed variables in an HMM may be discrete or continuous, and a variety of different conditional distributions can be used to model them. If both the latent and the observed variables are Gaussian (with a linear-Gaussian dependence of the conditional distributions on their parents), then we obtain the *linear dynamical system*.

2.2.3 Hidden Markov Models

Hidden Markov models (*HMM*) are used in different tasks and domains. They are the main tools in commercial applications in almost all the branches of speech processing, speech recognition, and speaker recognition. However, speech is not the only area dominated by HMMs, we can find their influence in handwriting recognition, machine translation, and sign language recognition. Although, HMMs benefit from relatively fast and powerful training and decoding algorithms, some of their assumptions limit their generality among other models [Bourlard and Morgan, 1994]. We can name these limiting factors as:

1. Poor discrimination. The models are trained independently one another. The algorithms use maximum likelihood inference which is a generative model.
2. *A priori* choice of model topology and statistical distributions. For instance, it is really common to use *Gaussian mixture models* (*GMM*) as the probability density functions of the random variables a priori.
3. Assumption that the state sequences are first-order Markov chains. For instance in some cases it may be possible the dependency between random variables is from higher orders or even it might be differ from one pair to another.
4. Sometimes in speech (where they are mainly used), no acoustical context is used, so that possible correlations between successive acoustic vectors is overlooked.

Note that there are different methods to circumvent these weaknesses of HMM such as using a combination with neural networks (hybrid models) which will be discussed later.

Hidden Markov models fall in a subclass of *dynamical Bayesian networks* which are themselves a kind of Bayesian networks discussed in section 2.2.1. We start with state space approach of Figure 2.11 for HMM and then unfold this illustration in order to extract the probabilistic graphical model representation.

We can then write the conditional distribution explicitly in the form

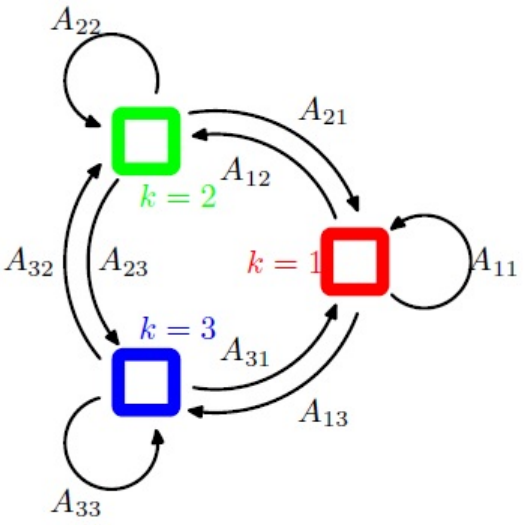


FIGURE 2.11: Transition diagram showing a model whose latent variables have three possible states corresponding to the three boxes. The black lines denote the elements of the transition matrix A_{jk} [Bishop, 2006].

$$p(\boldsymbol{z}_n|\boldsymbol{z}_{n-1}, \boldsymbol{A}) = \prod_{k=1}^{K}\prod_{j=1}^{K} A_{jk}^{z_{n-1,j}z_{nk}} \tag{2.11}$$

where $\boldsymbol{A}$ is a matrix each of which elements are known as *transition probabilities*, they are given by $A_{jk} \equiv p(z_{nk} = 1|z_{n-1,j} = 1)$, and because they are probabilities, they satisfy $0 \leqslant A_{jk} \leqslant 1$ with $\sum_k A_{jk} = 1$. The initial latent node $\boldsymbol{z}_1$ has no parent node, and so it has a marginal distribution $p(\boldsymbol{z}_1)$ represented by a vector of probabilities (initial probabilities) $\boldsymbol{\pi}$ with elements $\pi_k \equiv p(z_{1k} = 1)$, so that

$$p(\boldsymbol{z}_1|\boldsymbol{\pi}) = \prod_{k=1}^{K} \pi_k^{z_{1k}} \tag{2.12}$$

where $\sum_k \pi_k = 1$. Note that there is another representation for HMMs which is not a probabilistic graphical representation, see Figure 2.11 for the case of $K = 3$, here the boxes (circles are not used to highlight the differences) do not represent the random variables, they are states of a single random variable. We can also unfold the state diagram and sketch another diagram which is called *lattice* or *trellis* diagram and shows the state transitions over time, see Figure 2.12.

Similar to the general probabilistic graphical models, we have to define the conditional dependencies of the probabilistic densities over the hidden variables given the observed ones, $p(\boldsymbol{x}_n|\boldsymbol{z}_n, \boldsymbol{\phi})$, where $\boldsymbol{\phi}$ is a set of parameters governing the distribution which are called *emission* probabilities and might for example be given by Gaussians if the elements

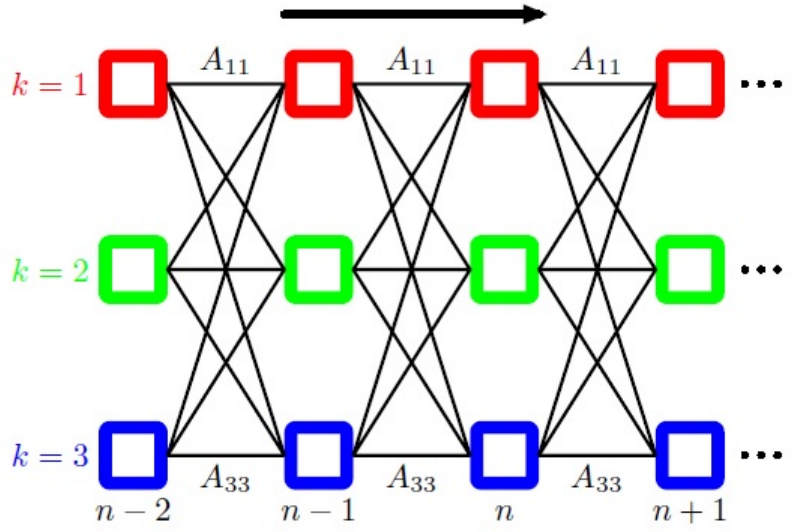


FIGURE 2.12: If we unfold the state transition diagram of Figure 2.11 over time, we obtain a lattice, or trellis, representation of the latent states. Each column of this diagram corresponds to one of the latent variables $\boldsymbol{z}_n$ [Bishop, 2006].

of $\boldsymbol{x}$ are continuous variables, or by conditional probability tables if $\boldsymbol{x}$ is discrete. Since $\boldsymbol{x}_n$ is observed, the distribution $p(\boldsymbol{x}_n|\boldsymbol{z}_n, \boldsymbol{\phi})$ consists, for a given value of $\boldsymbol{\phi}$, of a vector of K numbers corresponding to the K possible states of the binary vector $\boldsymbol{z}_n$. We can represent the emission probabilities in the form

$$p(\boldsymbol{x}_n|\boldsymbol{z}_n, \boldsymbol{\phi}) = \prod_{k=1}^{K} p(\boldsymbol{x}_n|\boldsymbol{\phi})^{z_{nk}} \tag{2.13}$$

Here in this section we will cope with *homogeneous* models. In this kind of models the parameters $\boldsymbol{A}$ are common among all the conditional densities and also all the emission distributions share the same parameters $\boldsymbol{\phi}$. The joint probability distribution over both latent and observed variables is then given by

$$p(\boldsymbol{X}, \boldsymbol{Z}|\boldsymbol{\theta}) = p(\boldsymbol{z}_1|\boldsymbol{\pi}) \left[\prod_{n=2}^{N} p(\boldsymbol{z}_n|\boldsymbol{z}_{n-1}, \boldsymbol{A}) \right] \prod_{m=1}^{N} p(\boldsymbol{x}_m|\boldsymbol{z}_m, \boldsymbol{\phi}) \tag{2.14}$$

where $\boldsymbol{X} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\}$, $\boldsymbol{Z} = \{\boldsymbol{z}_1, \ldots, \boldsymbol{z}_N\}$, and $\boldsymbol{\theta} = \{\boldsymbol{\pi}, \boldsymbol{A}, \boldsymbol{\phi}\}$ denotes the set of parameters governing the model. So far we have not discussed about the emission probabilities, we only factorized the joint probabilities. It is possible to adopt density models such as Gaussian or mixture of Gaussians (which are quite common in the field of speech), and even merge them with the neural networks which are discriminative models themselves. These can be used to model the emission density $p(\boldsymbol{x}|\boldsymbol{z})$ directly

(discriminative models), or to provide a representation for $p(\boldsymbol{z}|\boldsymbol{x})$ that can be converted into the required emission density $p(\boldsymbol{x}|\boldsymbol{z})$ using Bayes' theorem (generative models) [Bishop, 2006].

As I mentioned earlier the applications and the algorithms for decoding of HMMs and other probabilistic graphical models are quite wide, here I have tried to introduce some basics to have an overview and an insight over the subject.

Chapter 3

Introduction to Neural Network

The brain, the masterpiece of creation, is almost unknown to us, said Nicolaus Steno in 1669. In the fourth century B.C., Aristotle considered the brain to be a secondary organ that served as a cooling agent for the heart and a place in which spirit circulated freely. He designated the space in which all the spirits came together as the *sensus communis*- the origins of our much more metaphorical term, *common sense*. Aristotle had famously written, *There is nothing in the intellect that is not in the senses*. Far after that, at the time of writing this thesis, brain is still a complicated and semi-unknown structure which needs more research and investigations.

Human ability to recognize the patterns guides scientists to draw their inspiration from the ability of brain. In order to do so, we have to be familiar to the structure and functioning of the brain as a complex system. Here in this chapter we describe briefly the structure and simplified model of the mind and nervous system, and its motivations.

3.1 Basics of Neural Networks

How does the brain think? This is one of the most challenging unsolved questions in science [O'Reilly and Munakata, 2000]. [Bono, 1969] believes that there are those who suppose that the brain will forever remain a mystery and those who suppose that one day the way the brain works will be revealed in all its details, I am personally amongst those who try not to accept the idea of the first group. The nervous system is composed of two kinds of cells: *neurons* and *glia*. Only the neurons transmit impulses from one location to another [Kalat, 2009]. In this section we will introduce the neurons, as the elementary building blocks of the brain and the rest of the nervous system, and its simplified mathematical model, used in artificial neural networks.

3.1.1 What is a Neuron?

The neuron, say a unit in computational models, is the basic information-processing block for human cognition. The structure and the working mechanism of a neuron with all its details is almost impossible to simulate with even modern computers, consequently, usually scientists prefer to use simplified models of this atomic block. These simplifications also help at discovering which characteristics of neurons are most cognitively relevant [O'Reilly and Munakata, 2000]. A typical neuron is illustrated in Figure 3.1 with *nucleus* and all the *dendrites*. You can think of the dendrites as input to the system while the output can be transferred to the *synapses* via *axon*, and synapses are the junctions to other neurons. Roughly speaking there are lots of neurons in human brain, around 100 billion (10^{11}) each of which is connected to the average 7000 neurons. The received information in a specific neuron (unit), from many different sources (inputs), are integrated into a single real number. This real number is used to show how well this information matches what the neuron has become specialized to detect. After this evaluation, the neuron outputs something that illustrates the results. This is the famous *integrate-and-fire* model of neural function [O'Reilly and Munakata, 2000].

It seems that the brain is able to preform the processing task simultaneously (parallel) across billions of neurons, which are distributed throughout the brain, this is called the *parallel distributed processing* (*PDP*) [Rumelhart and McClelland, 1987; McClelland and Rumelhart, 1987]. This method is quite in contrast to the ordinary computers used in our daily routine. In a standard computer architecture memory and processing units are separated one from another. The processing is done in CPU, the information is retrieved from the memory units and after processing stage sent back to its original place.

The neurons are nondeterministic so what they do is they emit spikes depending on how much they are active, so even if you put the same input, the output at the other end is random to some extent, of course the more inputs that it gets excited the more spikes it will generate per second but is not exactly the same. They are also slow, it means that they often take like 1 mili second to react so this does not seem like a powerful computational machinery, yet we obviously have a 100 billion of them in our brain and they seem to be doing something right. One of the things if you look at the system theory point of view is the input and output relationship. you see a sigmoidal function.

so you have all the inputs and multiply them by weights and subtracts some threshold which is the minimum threshold that a neuron needs to get activated, when the inputs are above some value the neuron fires otherwise it is off. So it is on or off with some region in which it is changing. This is the main function that people have been using for neural networks.

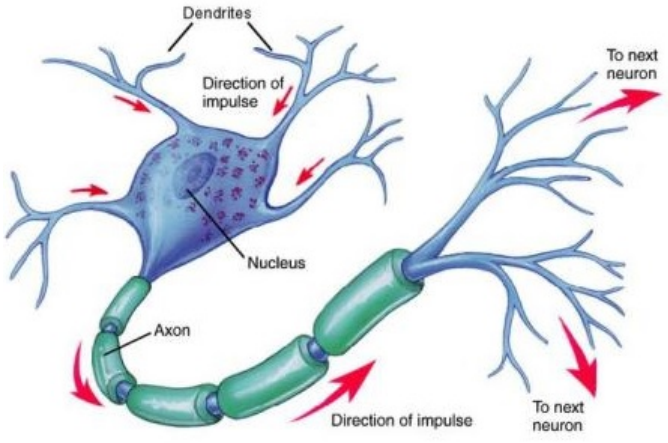


FIGURE 3.1: A Neuron.

In artificial neural networks, the information processing is done in two stages, one is a linear addition like what mentioned before for natural neurons, and the other is a non-linear function[1], which produces the actual output of the unit (neuron) [Anderson, 1995].

$$y = f(b + \sum_i x_i w_i) \tag{3.1}$$

where b is the bias, x_i is the ith input to the neuron, w_i is the weight of the ith input, $f(\cdot)$ is the non-linear function, and y is the output.

The Parallel Distributed Processing group in [Rumelhart and McClelland, 1987; McClelland and Rumelhart, 1987], referred to a function that compresses the range values of its input from $[-\infty, \infty]$ to a limited range as a *squashing function*. If the function is monotonic, differentiable, and smooth, it is often realized by a *sigmoid* or S-shaped curve, a term whose theoretical significance was pointed out by several in the 1970s [Wilson and Cowan, 1972; Grossberg, 1976]. One example of a commonly used sigmoid would be simple *logisitc* function[2]:

$$f(z) = \frac{1}{1 + e^{-z}} \tag{3.2}$$

[1] We can make an exception about the linear neurons, which there is only the first stage of information processing, however the taxonomy above is still met since it can be imagined as a special case of non-linear functions

[2] There are other sigmoid functions such as *arctangent*, *hyperbolic tangent*, *Gudermannian function*, *error function*, *generalized logistic function*, and some algebraic functions such as $f(x) = \frac{x}{\sqrt{1+x^2}}$. The integral of any smooth, positive, *bump-shaped* function will be sigmoidal, thus the *cumulative distribution functions* for many common probability distributions are sigmoidal, the most famous one is the error function, which is related to the cumulative distribution function of a normal distribution.

There is also another kind of non-linear function shown to be quite useful for modeling the output of a neuron. The *limit function*, which clips the values above and below some preset value, for instance *binary threshold neurons* are of this type.

$$y = \begin{cases} 1 & z \geq 0 \\ 0 & otherwise \end{cases} \tag{3.3}$$

where z is the summation of inputs and bias terms, $b + \sum_i x_i w_i$. There is also *rectified linear neurons*:

$$y = \begin{cases} z & z \geq 0 \\ 0 & otherwise \end{cases} \tag{3.4}$$

As shown in [Chapman, 1966], which is the input-output relation of a real neuron, it seems that binary threshold function, introduced before, is probably suitable to be considered as the non-linear function to model the evaluation process of the neuron. However, there are other options such as sigmoidal functions which might be even more suitable than the bianry threshold, see Figure 3.2. Note that these observed firing rates are the effects of several different processes and phenomena in a neural system, so it is hard to exploit a smooth, linear pattern. However, we can approximate this graph, achieved by experimental results, fairly enough with mathematical notations.

As mentioned before, neural mechanism is quite random, say noisy in nature, so it might be possible to interpret them as probabilistic functions of their inputs. Like any other dynamic system, they are characterized by transient and stationary responses to the stimuli. The transient response is quite dependent on time and initial values (states), however, the stationary part of the response is less dependent on time or even it is time-independent and could be interpret by a time independent probability distribution, like stationary processes. Notice that an exact description of the characteristics of a stochastic neuron, either transient or stationary, is not possible [Kappen, 2001].

The *stochastic binary neurons* use the same equations as the logistic units, equation 3.2. They compute a real value which is the probability that they will output as spike, however then instead of outputting that probability as a real number they actually make a probabilistic decision, so what they actually output is either 1 or 0.

$$p(S = 1) = \frac{1}{1 + e^{-z}} \tag{3.5}$$

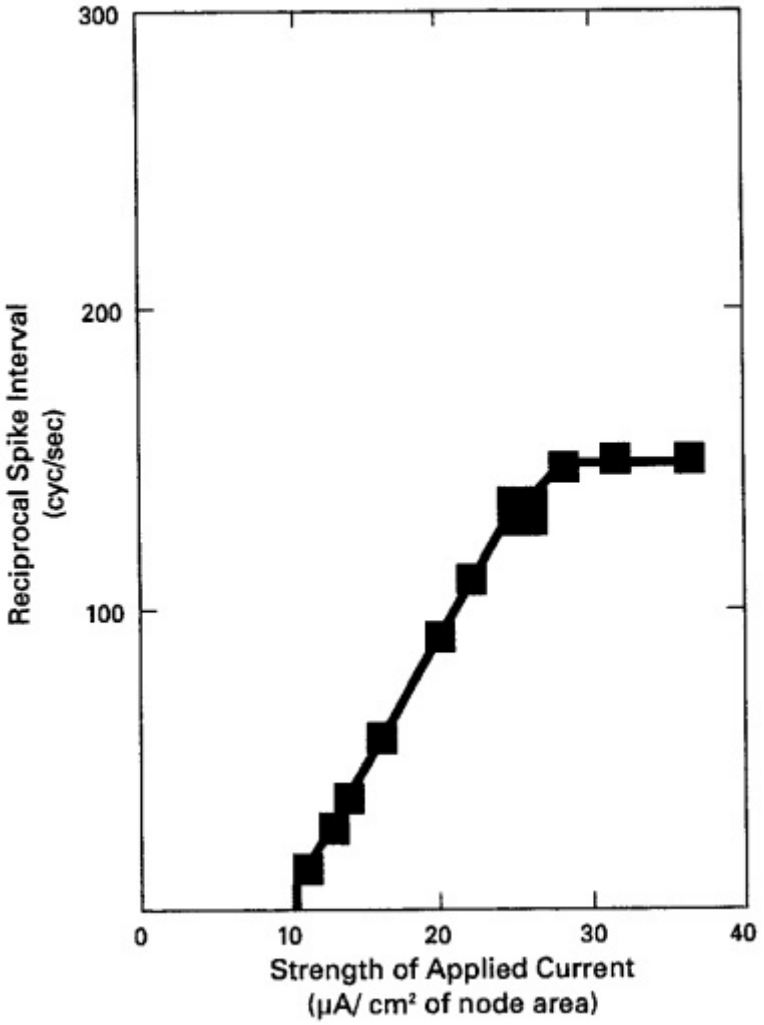


FIGURE 3.2: Experimental data relating stimulus intensity and firing rate from a crab motor neuron, a biological sigmoid, from [Chapman, 1966].

In other words they treat the output of the logistic as the probability of firing (producing a spike) in a short time window, $p(S = 1)$. Naturally if the input is very large and positive, they will almost always produce a 1 and if it large and negative they will produce a 0. We can do the similar trick for rectified linear units, we can say that the output, there is real-valued that comes out of a rectified linear unit, if its input is above zero, is the rate of producing spikes. So that is deterministic, however once we have figured out this rate of producing spikes, the actual times at which spikes are produced is a random process, it is a *Poisson* process. In summary the rectified linear unit determine the rate, but intrinsic randomness in the unit determines when the spikes are actually produced.

3.1.2 Neural Networks Architectures

The architecture of neural networks greatly depends on the characteristics and nature of the problem. Different architectures and neuron types have been used such as the *single-layer perceptron*, *multilayer perceptron* (*MLP*), *competitive networks*, *self-organizing map* (*SOM*), *recurrent networks*, and so on, which are illustrated in Figure 3.3

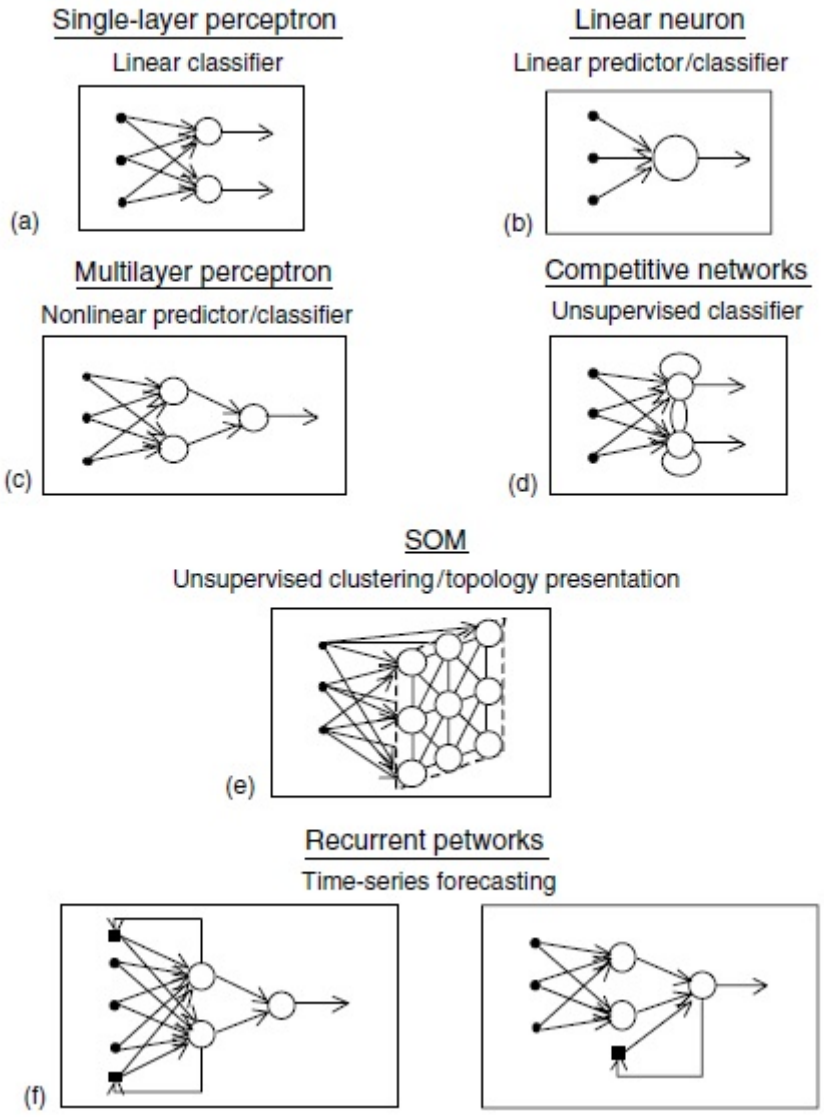
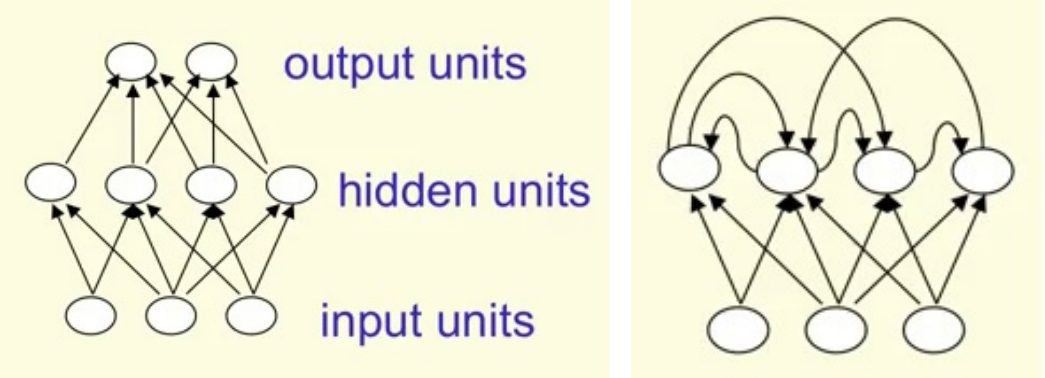


FIGURE 3.3: Some neural networks architectures for performing some specific tasks such as *presentation*, *classification*, and *prediction*. a) single-layer perceptron; b) linear neuron; c) multilayer perceptron (MLP); d) competitive networks; e) self-organizing feature map; f) recurrent networks [Samarasinghe, 2007].

One possible classification for neural networks is in terms of their architectures, the connection between different units (neurons) and different layers. There are three main types of neural networks a) Feed-forward neural networks; b) Recurrent neural networks; and c) Symmetrically connected neural networks.

The most common type of neural network is the feed-forward architecture. As you can see in Figure 3.4(a) the first layer is the input and the last layer is the output, and there might be one or more layers of hidden units in between. If there is more than one hidden layer, it is called *deep* neural network. These networks compute a series of transformations between their input and output. So at each layer, you get a new representation of the input in which things that were similar in the previous layer may have become less similar, or things that were dissimilar before may have become more similar. For example in speech recognition, we would like the same thing said by different speakers to become more similar, and different things said by the same speaker to become less similar as going up through the layers of the network. In order to achieve this, we need the activities of the neurons in each layer to be a non-linear function of the activities of the neurons in the previous layer.



(a) Feed-forward neural network (b) Recurrent neural network

FIGURE 3.4: Classification of neural networks architectures; a) Feed-forward neural network; b) Recurrent neural network.

Recurrent neural networks are more powerful than the feed-forward networks. As depicted in Figure 3.4(b), they have directed cycles in their connection graph. what it means is that if you start at a node and you follow the arrows, you can sometimes get back to the neuron you started at. They can have very complicated dynamics, and this can make them very difficult to train. They are more biologically realistic. Recurrent networks are a very natural way to sequential data as we will see later.

The symmetrically connected networks are like recurrent neural networks, but the connections between units are symmetrical (they have the same weight in both directions). Symmetrically connected nets without hidden units are called *Hopfield nets*, one of the simplest kinds of *Energy-based* models.

3.2 Neural Networks in Pattern Recognition

As discussed in Chapter 2, first important step for building an intelligent system is representation. One of the main issues in representation is model selection in order to describe and explain the interactions between different *things* and/or *phenomena* within a system or framework. Neural Network is one of such models that characterizes the interactions within a system (or pattern) in terms of combination of parametric basis functions [Bishop, 2006].

The artificial neural network, say neural network hereafter, is nothing except a complex of interconnected units try to extract (learn) information (trends in complex data) from the data [Samarasinghe, 2007]. They are able to be used in variety of tasks such as prediction or function approximation, pattern classification and recognition, probability density function estimation, feature extraction and dimension reduction, clustering, forecasting, association or pattern matching, time-series analysis, non-linear system modeling or inverse modeling, multivariate function approximation or mapping, optimization, intelligent control and signal processing, some of these applications are shown in Figure 3.5 [Zaknich, 2003].

Despite all the things mentioned in section 3.1, our focus in this section is therefore on neural networks as efficient models for statistical pattern recognition.

3.2.1 Neural Networks and Probabilistic Representation and Inference

As seen in subsection 2.1.2, we can do the recognition task by means of different class of methods. Here we start with the easiest case, discriminant function[1], and continue to provide a probabilistic interpretation to the network outputs, as we have seen the advantages of such representation in subsection 2.1.2.

Note that the architecture itself has no sense about the meaning of the outputs, this is our task, using a proper *activation function*, suitable number of layers and units in each layer, training type and algorithm, to induce a desired interpretation. For example for the case of probabilistic interpretation, we have to use functions such that they meet the probability function constraints at the output layer:

1. The probabilities must be non-negative.
2. The probabilities must be between $[0, 1]$.
3. The summation of all output probabilities must be equal to one.

[1] A discriminant is a function that takes an input vector $\boldsymbol{x}$ and assigns it to one of k classes, denoted $\mathcal{C}_k$ [Bishop, 2006].

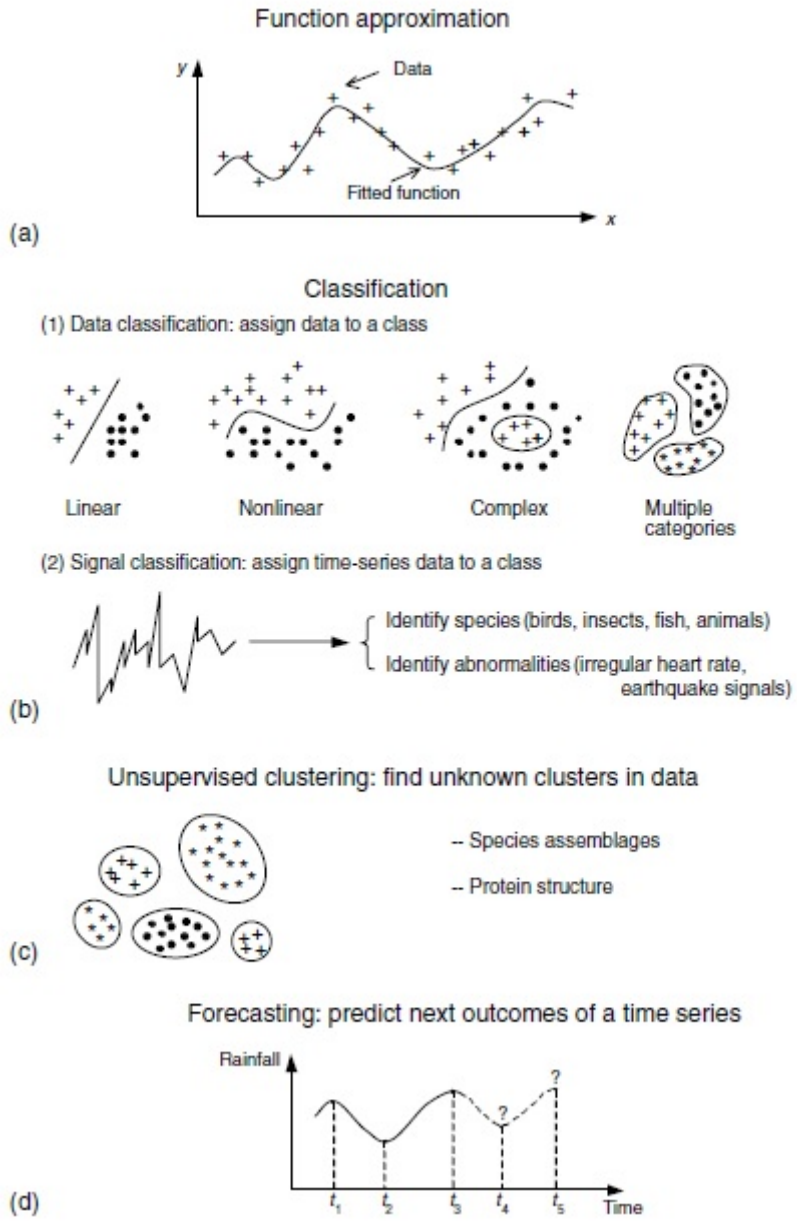


FIGURE 3.5: Some applications of neural networks. a) fitting models to data; b) complex classification tasks; c) discovering clusters in data; d) time-series forecasting [Samarasinghe, 2007].

3.2.2 Training in Neural Networks (Learning)

As mentioned in subsection 2.1.3, there are different classes of learning methods. All learning paradigms including supervised, unsupervised and reinforcement learning, result in an adjustment of the weights of the connections between units, according to some modification rule. Virtually all learning rules for models of this type can be considered as a variant of the Hebbian learning rule. The basic idea is that if two units j and k are active simultaneously, their interconnection must be strengthened. If j receives input from k, the simplest version of Hebbian learning prescribes to modify the weight w_{jk} with:

$$\Delta w_{jk} = \gamma y_j y_k \tag{3.6}$$

where γ is a positive constant of proportionality representing the *learning rate*. Another common rule uses not the actual activation of unit k but the difference between the actual and desired activation for adjusting the weights:

$$\Delta w_{jk} = \gamma y_j (t_k - y_k) \tag{3.7}$$

in which t_k is the desired (target) activation provided by a teacher. This is often called the *Widrow-Hoff rule* or the *delta rule* [Kröse and Van der Smaft, 1996].

There are different algorithms for different architectures of neural networks, mainly dealing with the problem optimization of a cost function. In the later chapters we will be familiarized with some of these algorithms such as back-propagation, contrastive divergence, back-propagation through time and so on.

3.3 Energy-Based Neural Networks

In general neural networks are not probabilistic graphical models, however Boltzmann machines, among the new generation neural networks, are closely related to undirected graphical models (Markov random fields). In this section we are dealing with the concept of *energy-based* models, which are the building blocks for more sophisticated architectures, to be discussed in later chapters.

We will use the background, introduced in subsection 2.2.1, to establish a formalism for one the heirs to the probabilistic graphical models.

3.3.1 Boltzmann Machine

Stochastic Hopefield nets with hidden units, which is also called Boltzmann machines are useful at modeling binary data. Given a training set of binary vectors, fit a model that will assign a probability to every possible binary vector. For several reasons it might be important to do such task:

- This is useful for deciding if other binary vectors come from the same distribution (e.g., documents represented by binary features that represents the occurrence of a particular word).
- It can be used for monitoring complex systems to detect unusual behavior.
- If we have models of several different distributions it can be used to compute the posterior probability that a particular distribution produced the observed data.

The most natural way to think about generating a binary vector is to first generate the states of some latent variables, then use the latent variables to generate the binary vectors. In a causal model the data is generated in two sequential steps:

1. Pick the hidden states from their prior distribution.
2. Pick the visible states from their conditional distribution given the hidden states.

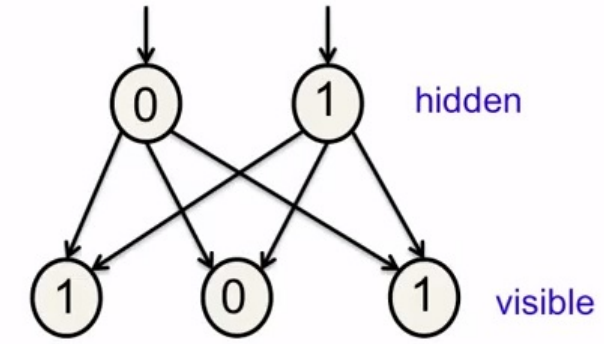


FIGURE 3.6: A causal generative neural network model.

Figure 3.6 is a kind of neural network, which is a causal generative model. It uses logistic units and biases for the hidden units and weights on the connections between hidden and visible units to assign a probability to every possible visible vector. The probability of generating a particular vector $\boldsymbol{v}$, is computed by summing over all possible hidden states. Each hidden state is an *explanation* of vector $\boldsymbol{v}$, mathematically interpreted in 3.8. This is the probability of a causal model, and it is probably the most natural way

to think about generating data, even in some literature the term generated model and causal models are equivalent.

$$p(\boldsymbol{v}) = \sum_{\boldsymbol{h}} p(\boldsymbol{h})p(\boldsymbol{v}|\boldsymbol{h}) \tag{3.8}$$

There is a completely different kind of model, Energy-based models, such as *Boltzmann machines*. In this kind of model, the data is not generated causally. Instead everything is defined in terms of the energies of joint configurations of the visible and hidden units. The energies of joint configurations are related to their probabilities in two ways. One way is to simply define the probability to be the probability of a joint configuration of the visible and hidden variables proportional to a exponential term in 3.9 or alternatively, we can define the probability to be the probability of finding the network in that joint configuration after we have updated all of the stochastic binary units many times.

$$p(\boldsymbol{v}, \boldsymbol{h}) \propto e^{-E(\boldsymbol{v},\boldsymbol{h})} \tag{3.9}$$

As you can see the energy of a joint configuration is:

$$-E(\boldsymbol{v}, \boldsymbol{h}) = \sum_{i \in visible} v_i b_i + \sum_{k \in hidden} h_k b_k + \sum_{i<j} v_i v_j w_{ij} + \sum_{i,k} v_i h_k w_{ik} + \sum_{k<l} h_k h_l w_{kl} \tag{3.10}$$

where b_i is the bias of unit i, and w_{ij} is the weight between unit i and j. For the visible-visible and hidden-hidden interactions, i and k is assumed to be less than j and l respectively, $i < j$, $k < l$ in order to avoid counting the interaction of a unit with itself and also avoid counting a pair twice. Now the probability of a joint configuration over both visible and hidden units depends on the energy of that joint configuration compared with the energy of all other joint configurations, see 3.11.

$$p(\boldsymbol{v}, \boldsymbol{h}) = \frac{e^{-E(\boldsymbol{v},\boldsymbol{h})}}{\sum_{\boldsymbol{u},\boldsymbol{g}} e^{-E(\boldsymbol{u},\boldsymbol{g})}} \tag{3.11}$$

The probability of a configuration of the visible units is the sum of the probabilities of all the joint configurations that contain it, see 3.12.

$$p(\boldsymbol{v}) = \frac{\sum_{\boldsymbol{h}} e^{-E(\boldsymbol{v},\boldsymbol{h})}}{\sum_{\boldsymbol{u},\boldsymbol{g}} e^{-E(\boldsymbol{u},\boldsymbol{g})}} \tag{3.12}$$

3.3.2 Restricted Boltzmann Machine

A Restricted Boltzmann Machine (RBM) is an undirected graphical model with a set of binary visible variables $\boldsymbol{v}$, a set of binary latent (hidden) variables $\boldsymbol{h}$, and a weight matrix $\boldsymbol{W}$ for bipartite connections between $\boldsymbol{v}$ and $\boldsymbol{h}$. The probability of an RBM configuration is given by

$$p(\boldsymbol{v}, \boldsymbol{h}) = \frac{1}{Z} e^{-E(\boldsymbol{v}, \boldsymbol{h})} \tag{3.13}$$

where Z is the partition function and E is the energy function defined as

$$-E(\boldsymbol{v}, \boldsymbol{h}) = \sum_i a_i v_i + \sum_j b_j h_j + \sum_{i,j} v_i h_j w_{ij} \tag{3.14}$$

where a_i and b_j are biases for corresponding visible and latent variables respectively, and w_{ij} is the symmetric weight between v_i and h_j. Given the visible variables, the latent variables are conditionally independent of each other, and vice versa [Garg and Henderson, 2011]. As you can see from 3.14, it is a Boltzmann machine without any hidden-hidden and visible-visible interactions.

Now with an example, Figure 3.7, we would be familiar to how we compute the probabilities regarding a simple RBM. The first thing we do is to write down all possible states of visible units. For each state of visible units there are four possible states of the hidden units that could go with it. So that gives us 16 possible joint configurations.

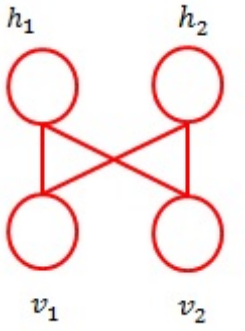


FIGURE 3.7: An example of Restricted Boltzmann Machine and how weights define a distribution.

The value of weight for $h_1 - v_1$ connection is 2, for $h_1 - v_2$ is -0.5, for $h_2 - v_1$ is -1, and for $h_2 - v_2$ is 1, also assume that we have no bias term. Now for each of those joint configurations, we are going to compute its negative energy $-E$. We then take the negative energies and we exponentiate them, and that would give us unnormalized

probabilities. If we add all those up to 32.42 and then we divide everything by 32.42, we get the probabilities of joint configurations. Now, if we want the probability of a particular visible configuration, we have to sum over all the hidden configurations that could go with it, see Table 3.1.

$\boldsymbol{v}$	$\boldsymbol{h}$	$-E$	e^{-E}	$p(\boldsymbol{v},\boldsymbol{h})$	$p(\boldsymbol{v})$
11	11	$\frac{3}{2}$	4.48	0.14	
11	10	$\frac{3}{2}$	4.48	0.14	
11	01	0	1	0.03	0.34
11	00	0	1	0.03	
10	11	1	2.72	0.08	
10	10	2	7.39	0.23	
10	01	-1	0.37	0.01	0.35
10	00	0	1	0.03	
01	11	$\frac{1}{2}$	1.65	0.05	
01	10	$-\frac{1}{2}$	0.61	0.02	
01	01	1	2.72	0.08	0.18
01	00	0	1	0.03	
00	11	0	1	0.03	
00	10	0	1	0.03	
00	01	0	1	0.03	0.12
00	00	0	1	0.03	

TABLE 3.1: An example of computing the joint probability of a Restricted Boltzmann Machine.

There might be difficulties when the model is bigger than our example. Obviously, in the network we just computed, we can figure out the probability of everything due to the size of the model, however when we face a very large model we cannot do these exponentially large amount of computations. If there is more than a few hidden units, we cannot actually compute that partition function, because has exponentially many terms. In order to circumvent this problem, we may use *Markov Chain Monte Carlo* to get samples from the model starting from a random global configuration and then keep picking units at random and allowing them to stochastically update their states based on their energy gaps. Those energy gaps being determined by the states of all the other units in the network. If we keep doing that until the Markov chain reaches the stationary distribution (*thermal equilibrium* at a temperature of 1), then we have a sample from the model, and we can compute the probability of the global configuration, $p(\boldsymbol{v},\boldsymbol{h})$.

In order to compute the posterior probability, $p(\boldsymbol{h}|\boldsymbol{v})$, we have to use sampling from the posterior distribution over hidden configurations for a given data vector. It is similar to get a sample from the model, except that we keep the visible units clamped to the data vector we are interested in. Note that this time only hidden units are allowed to change states. The reason we need to get samples from the posterior distribution, given a data vector, is we might want to know a good explanation for the observed data, and we might want to base our actions on that good explanation, however we also need that for learning the weights.

The Boltzmann machine learning algorithm is an unsupervised learning algorithm, unlike the typical user back-propagation, where we have an input vector and we provide it with a desired output, in Boltzmann machine learning we just give it the input vector, there are no labels. What the algorithm is trying to do is to build a model of a set of input vectors, though it might be better to think of them as output vectors. What we want to do is maximizing the product of the probabilities, that the Boltzmann machine assigns to a set of binary vectors. The ones in the training set. This is equivalent to maximizing the sum of the log probabilities that the Boltzmann machine assigns to the training vectors. It is also equivalent to maximizing the probability that we would obtain exactly the N training cases if we did the following way:

1. Let the network settles to its stationary distribution N different times with no external input.

2. Sample the visible vector once each time.

It is an iterative procedure, after we sample the visible vector once, then we let it settles again, and sample the visible vector again, and so on. With this learning algorithm we learn the parameters that define a distribution over the visible vectors.

The learning could be difficult. If you consider a chain of hidden units with visible units attached to the two ends see Figure 3.8, and if we use a training set that consists of $(1, 0)$ and $(0, 1)$, we want the product of all the weights to be negative. In other words we want the two visible units to be in opposite states,then we can achieve that by making sure that the product of all those weights is negative. What this means is if we are thinking about learning weight W_1, we need to know other weights.This is probably the most important reason for the difficulty of learning algorithms with Boltzmann machines.

To circumvent this problem, there is a simple learning algorithm. Everything that one weight needs to know about the other weights and about the data, is contained in the difference of two correlations.

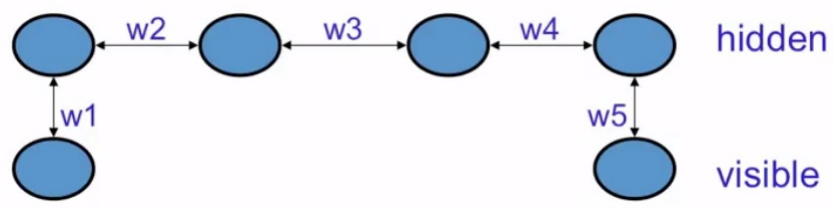


FIGURE 3.8: A chain of hidden units with the visible units at the two ends.

$$\frac{\partial \log p(\boldsymbol{v})}{\partial w_{ij}} = \langle s_i s_j \rangle_{\boldsymbol{v}} - \langle s_i s_j \rangle_{model} \tag{3.15}$$

where $\langle \cdot \rangle_{\boldsymbol{v}}$ is the expected value of product of states at thermal equilibrium when $\boldsymbol{v}$ is clamped on the visible units, and $\langle \cdot \rangle_{model}$ is expected value of product of states at thermal equilibrium with no clamping. Since the derivative of the log probability of a visible vector is this simple difference of correlations we can make the change in the weight be proportional to the expected product of the activities average over all visible vectors in the training set, we call *data* subtracted by a term which is the product of the same two activities when you are not clamping anything, and the network has reached the thermal equilibrium with no external interference.

$$\Delta w_{ij} \propto \langle s_i s_j \rangle_{data} - \langle s_i s_j \rangle_{model} \tag{3.16}$$

The first term in the learning rule says raise the weights in proportion to the product of the activities the units have, when you are presenting data. This is the simplest form of what is known as a *Hebbian learning rule*. Donald Hebb, a long time ago suggested that synapses in the brain might use a rule like that. However, if you just use that rule, the synapses strengths will keep getting stronger, and the weights will all become very positive, and the whole system will blow up. We have to keep things under control, and this learning algorithm is keeping things under control by using that second term. It is reducing the weights in proportion to how often those two units are on together, while you are sampling from the model's distribution. We can also think of this as the first term is like the storage term for the *Hopfield Net*, and the second term is like the term for getting rid of spurious minima, this is in fact the correct way to think about it. This rule tells us exactly how much unlearning to do.

There is one obvious question, *why is the derivative so simple?*. The probability of a global configuration at thermal equilibrium, is an exponential function of its energy. So when we settle to thermal equilibrium we achieve a linear relationship between the log probability and the energy function. Now the energy function is linear in the weights, so we have a linear relationship between the weights and the log probability, and since we

are trying to manipulate log probabilities by manipulating weights, that is a log linear model. In fact, the relationship is very simple.

$$-\frac{\partial E}{\partial w_{ij}} = s_i s_j \tag{3.17}$$

It is the derivative of the energy with respect to a particular weight w_{ij}, is just the product of the two activities that weight connects. The process of settling to thermal equilibrium, propagates information about the weights. We do not need any explicit back propagation stage, we do need two stages. We need to settle with the data, and we need to settle with no data. Notice that the network behaving in the same way in those two phases. The unit, deep within the network, is doing the same thing, just with different boundary conditions. With back propagation the forward pass and the backward pass are really rather different. Another question is, *what is that negative phase for?*. It is like the unlearning in Hopfield net to get rid of the spurious minima. The marginal probability over visible vector is given by

$$p(\boldsymbol{v}) = \frac{\sum_{\boldsymbol{h}} e^{-E(\boldsymbol{v},\boldsymbol{h})}}{\sum_{\boldsymbol{u}} \sum_{\boldsymbol{g}} e^{-E(\boldsymbol{u},\boldsymbol{g})}} \tag{3.18}$$

what the top term in the learning rule is doing, is decreasing the energy of terms in that sum that are already large and it finds those terms by settling to thermal equilibrium with the vector $\boldsymbol{v}$ clamped so that it can find an $\boldsymbol{h}$ that goes nicely with $\boldsymbol{v}$, giving a nice low energy with $\boldsymbol{v}$. Having sampled those vectors $\boldsymbol{h}$, it then changes the weights to make that energy even lower. The second phase in the learning, the denominator, is doing the same thing for the partition function. It is finding the global configurations, combinations of visible and hidden states that give low energy, and therefore are large contributors to the partition function. Having find those global configurations, it tries to raise their energy so that they can contribute less. So the first term is making the top line big, and the second term is making the bottom line small.

Now in order to run this learning rule, we need to collect those statistics. We need to collect what are called the *positive statistics*, those are the ones when we have data clamped on the visible units, and also the negative statistics, those are the ones when we don't have the data clamped and that we are going to use for unlearning. An inefficient way to track these statistics was suggested by [Hinton and Sejnowski, 1983], and the idea is, in the positive phase we clamp a data vector on the visible units, setting the hidden units to random binary states. Then we keep updating the hidden units in the network, one unit at a time, until the network reaches thermal equilibrium at a temperature of one, starting at a high temperature and reducing it. Once we reach the thermal equilibrium,

we sample how often two units are on together $\langle s_i s_j \rangle$. In fact we are measuring the correlation of i and j with that visible vector clamped. We then repeat that over all the visible vectors, so that the correlation we are sampling is averaged over all the data. Then in the negative phase we don't clamp anything. The network is free from external interference. So, we set all of the units, both visible and hidden, to random binary states. And then you update the units, one at a time, until the network reaches thermal equilibrium, at a temperature of one, such that we did in positive phase. And again, we sample the correlation of every pair of units i and j, and we repeat that many times. It is very difficult to know how many times you need to repeat it, however certainly in the negative phase we expect the energy landscape to have many different minima, but are fairly separated and have about the same energy. The reason we expect that, is we are going to be using Boltzmann machines to do things like model a set of images, and we expect there to be reasonable images, all of which have about the same energy and then very unreasonable images, which have much higher energy, and so we expect a small fraction of the space to be these low energy states, and a very large fraction of space to be these bad high energy states. If we have multiple modes, it is very unclear how many times we need to repeat this process to be able to sample those modes.

Restricted Boltzmann machines have a much simplified architecture in which there are no connections between hidden units. This makes it very easy to get the equilibrium distribution of the hidden units if the visible units are given. That is once we clamped the data vector on the visible units, the equilibrium distribution of the hidden units can be computed exactly in one step, because they are all independent of one another, given the states of the visible units. The proper Boltzmann machine learning is still slow for a restricted Boltzmann machine however in 1998, Hinton discovered a very surprising shortcut that leads to the first efficient learning algorithm for Boltzmann machines. Even though this algorithm has theoretical problems, it works quite well in practice, and it let to revival of interests in Boltzmann machine learning.

In restricted Boltzmann machine, as we said before, we restrict the connectivity in order to make both inference and learning easier. It has only one layer of hidden units with any interconnection. There is also no connection between visible units, see Figure 3.9. The computer scientists call it a bipartite graph, there are two pieces, and within each piece there is no connection. In an RBM it only takes one step to reach the thermal equilibrium when the visible units are clamped. That means with a data vector clamped, we can quickly compute the expected value $\langle v_i h_j \rangle_{\boldsymbol{v}}$, because we can compute the exact probability with each j will turn on, and that is independent of all the other units in the hidden layer.

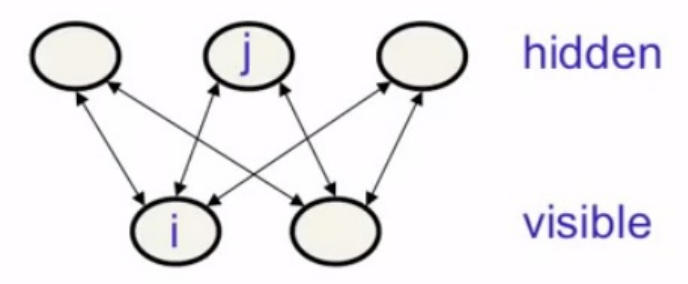


FIGURE 3.9: A simple Restricted Boltzmann Machine (RBM) with one hidden layer.

The probability that j will turn on is just the logistic function of the input that it gets from the visible units and quite independent of what other hidden units are doing. So, we can compute that probability all in parallel and that is a tremendous win. If we want to make a good model of a set of binary vectors, then the right algorithm to use for a restricted Boltzmann machine is one introduced by Tieleman in 2008, that is based on earlier work by Neal. In the positive phase, we clamp the data vector on the visible units, we then compute the exact value of $\langle v_i h_j \rangle$ for all pairs of a visible and a hidden unit. And for every connected pair of units, we average the expected value $\langle v_i h_j \rangle$ over all data in the mini-batch. For the negative phase, we keep a set of *fantasy particles*. Each particle has a value that is a global configuration. Then we update each fantasy particle a few times using alternative parallel updates. After each weight update, we update the fantasy particles a little bit, and that should bring them back too close to equilibrium. For every connected pair of units, average $v_i h_j$ over all the fantasy particles, this gives us the negative statistics. This algorithm actually works very well, and allows RBMs to build good density models or sets of binary vectors.

Now we start with a picture of an inefficient learning algorithm for an RBM, see Figure 3.10. We start by clamping a data vector on the visible units, and we call that time $t = 0$. So, we use times now, not to denote weight updates, but to denote steps in a Markov chain. Given that visible vector, we now update the hidden units. So we choose binary states for the hidden units and we measure the expected value, $\langle v_i h_j \rangle$, for all pairs of visible and binary units that are connected. We call that $\langle v_i h_j \rangle^0$ to indicate that it is measured at time zero. With the hidden units being determined by the visible units, and of course, we can update all the hidden units in parallel. Then we use the hidden vector to update all the visible units in parallel, and again we update all the hidden units in parallel. So, the visible vector $t = 1$, we call a reconstruction, or a one step reconstruction. We can keep going with the alternating chain that way. Updating visible units, and then hidden units, each set being updated in parallel. After we have gone for a long time we get to some state of the visible units, or we call $\langle v_i h_j \rangle^\infty$ to indicate it needs to be a long time and the system will be at the thermal equilibrium.

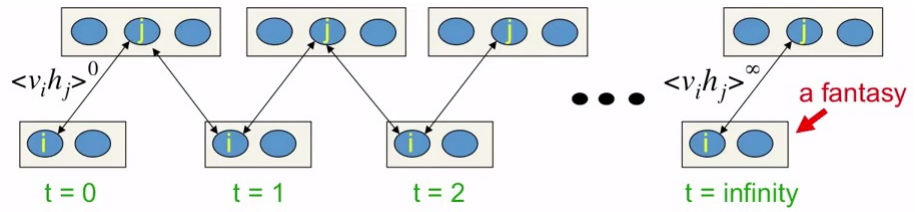


FIGURE 3.10: An example of Restricted Boltzmann Machine and how weights define a distribution. This is an inefficient learning algorithm for an RBM.

Now we can measure the correlation of v_i and h_j after the chains run for a long time, and we call it $\langle v_i h_j \rangle^\infty$. The visible state we have after a long time, we call it fantasy. So the learning rule is simply, we change w_{ij} by the learning rate times the difference between $v_i h_j$ at time $t = 0$ and $v_i h_j$ at time $t = \infty$. Of course, the problem with this algorithm is that we have to run this chain for a long time before it reaches thermal equilibrium, otherwise the learning may go wrong. In fact, the last statement is very misleading, it turns out even if we only run the chain for short, the learning still works.

$$\Delta w_{ij} = \epsilon(\langle v_i h_j \rangle^0 - \langle v_i h_j \rangle^\infty) \tag{3.19}$$

Contrastive divergence is the very surprising shortcut. We just run the chain up, down, and up again so, from the data, we generate a hidden state and from that we generate a reconstruction, and from that we generate another hidden state. We measure the statistics once we have done that. So, instead of using the statistics measured at equilibrium, we are using the statistics measured after doing one full update of the Markov chain. The learning rule is the same as before, except this much quicker to compute, and this clearly is not doing the maximum likelihood learning because, the term we are using for negative statistics is wrong. However, nevertheless, works quite well.

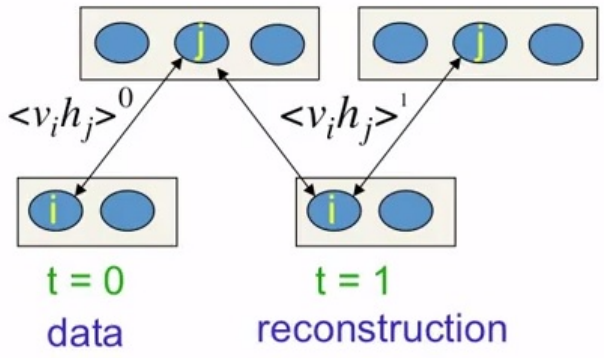


FIGURE 3.11: Contrstive divergence algorithm.

Now we explain why this shortcut works. If we start at the data, the Markov chain wanders away from the data and towards its equilibrium distribution, that is towards

things that is initial weights like more than the data. We can see what direction it is wandering in after only a few steps. If we know that the initial weights are very good, it is a waste of time to let it go all the way to equilibrium. We know how to change them to stop it wandering away from the data without going all the way to equilibrium.

$$\Delta w_{ij} = \epsilon(\langle v_i h_j \rangle^0 - \langle v_i h_j \rangle^1) \tag{3.20}$$

All we need to do is to lower the probability of the reconstructions of confabulations as a psychologist would call them, it produces after one full step, and then, raise the probability of the data. That will stop it wandering away from the data. Once the data and the places it goes to, after one full step, have the same distribution, then the learning will stop. Here is a picture of what is going on. Figure 3.12 is the energy surface in the space of global configurations. There is a data point on the energy surface, the data point is both visible vector and the particular hidden vector that we got by stochastic updating the hidden units. So, that hidden vector is a function of what the data point is. So, starting at that data point, we run the Markov chain for one full step to get a new visible vector and the hidden vector that goes with it. So, a reconstruction of the data point plus the hidden vector that goes with that reconstruction.

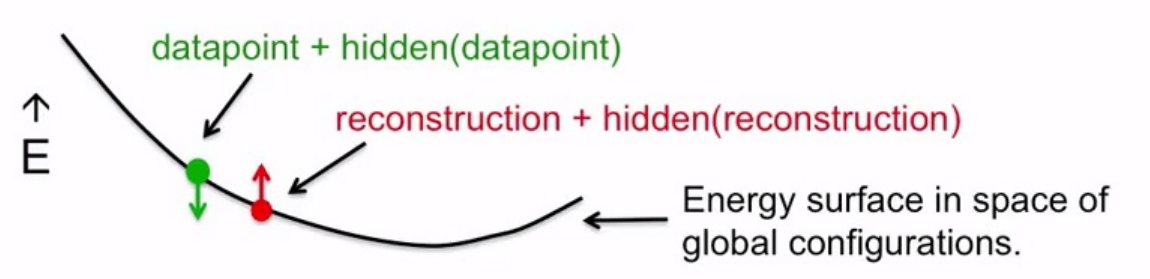


FIGURE 3.12: Energy surface in the space of global configuration.

We then change the weights to pull the energy down at the data point, and to pull the energy up at the reconstruction. The effect of that would be to make the surface looks like the Figure 3.13, and we will notice that we are beginning to construct an energy minimum at the data. We will also notice that far away from the data, things have stayed pretty much as they were before. So this shortcut of only doing one full step to get the reconstruction fails for places that are far away from the data point. We need to worry about the regions of the data-space that the model likes but which are very far from any data. These low energy holes cause the normalization term to be big and we cannot sense them if we use the shortcut. If we use persistent particles, where we remembered their states, and after each update, we updated them a few more times, then they would eventually find these holes. They would move into the holes, and the learning would cause the holes to fill up.

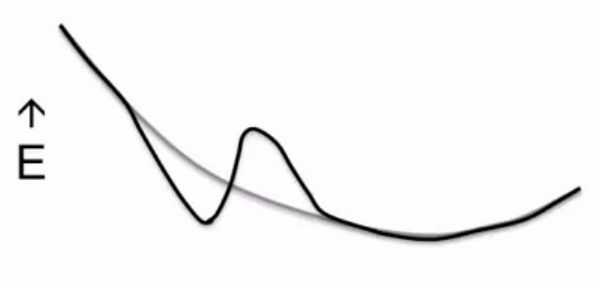


FIGURE 3.13: Reconstruction on energy surface in Contrastive Divergence. Changing the weights to pull the energy down at the data point, and pull it up at the reconstruction point.

A good compromise between speed and correctness is to start with small weights and to use *CD1*, that is contrastive divergence with one full step to get the negative data. Once the weights have grown a bit, the Markov chain is mixing more slowly, and now we can use *CD3*, and once the weights have grown more, we can use *CD5*, or nine or ten. So, by increasing the number of steps as the weights grow, we can keep the learning working reasonably well, even though the mixing rate of the Markov chain is going down.

Chapter 4

Deep Learning

In recent years we have witnessed a resurgence, say renaissance, in the field of neural networks after the second *winter* which had been started in 1990s. It is based on the representation of the data through multiple layers. This topic, which is referred to as *deep learning*, has been interacted and closely related to the existing research fields such as neural networks, graphical models, feature learning, unsupervised learning, optimization, pattern recognition, and signal processing. This is also motivated by neuroscience, more similarities to our understanding of the brain's intelligence (learning from unlabeled data), and there are already number of applications in computer vision, speech recognition, natural language processing, hand writing recognition and so on [Bengio et al., 2013; Laserson, 2011].

All the previous chapters are the preliminaries for this one. *Deep learning* is one of the progressive and promising area in machine learning for the future tasks involved in machine learning especially in the area of neural networks. We first introduce the concept and philosophy behind the *deep learning* and after that continue with the more technical issues such as different architectures and training methods. Note that our major attention is to the architectures and algorithms for sequential patterns (data), however, we also introduce the cases used in non-sequential (static) data, since it is sometimes an easy task to extend some of the methods for sequential applications.

4.1 Deep Learning Philosophy

As discussed in chapter 2, representation is one of the most important tasks related to pattern recognition. Each phenomenon, in order to be analyzed, must be firstly represented. This is an important issue since how information is presented can sometimes greatly impact on what we can do with it [Marr, 1982].

There are some reasons for increasing popularity of deep learning, which are both related to hardware and software developments, such as processing abilities by means of for example GPU units for the hardware and new machine learning or signal and information processing algorithms (either inference or learning) for the software [Deng, 2013].

As in [Deng, 2013], deep learning is a hierarchical method of representing data. The features are not complex in a point, they are distributed in different layers and different units one layer after another, this is the main task of deep learning, the one that distinguishes deep architectures from others [Deng, 2013]. Deep learning is exploited in representation learning, actually it is a branch of a category of machine learning methods based on learning representations.

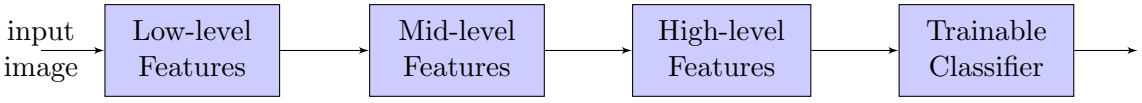


FIGURE 4.1: A block diagram of deep features. An architecture is deep once comprises more than one layer (stage) of non-linear features [Lecun and Ranzato, 2013].

As we can see in Figure 4.1, going through the stages, the level of abstraction is increased. For instance in image recognition we might start with the pixels as the lowest level and continue to edges, texton, motif, part and finally the object. We have the same structure in text recognition, from characters to more abstract levels such as words, word groups, clauses, sentences, to the total story [Lecun and Ranzato, 2013]. For example in *machine translation*, multiple layers of abstractization such as words, lemma, part-of-speech, and morphology [Koehn, 2009]. In speech multiple steps might be started from samples of the signal, spectral bands, sounds, up to (multiple layers in between) phone, phoneme, and word, however we can continue this abstractization to word gropus or clauses and sentence [Lecun and Ranzato, 2013].

The deep learning philosophy is fed by the assumption that the *truth* is not a solid monolithic *thing*, but rather a distributed concept flows along and among variety of *things*, *phenomena*, *categories* or here *units* and *layers*, which might all be integrated to superficially form a solid *entity*, *event*, *being*. One (human or machine) might not realize the *whole*, or even sometimes a part of it, without trying to perceive the overall structure and interactions between these single and monadic *elements*. Sometimes these individual *elements* have no meaning , *each within itself*, to us, however these are the individuals that form a *complex*, a *whole*.

During my research have realized the term deep learning is different from *deep neural network*, however the most important advances in deep learning are in the field of neural networks, that is why they are usually used as equivalent terms. Although the

neural networks are still the mainstream, there are methods, such as that of *deep kernel machines* introduce in subsection 4.2.11, which are not neural networks architectures [Lauzon, 2012].

4.2 Deep Learning Architectures

In this section and following subsections we will talk about more practical issues of deep learning such as architectures, inference ability and training algorithms. Whenever it is possible I may refer to some practical experiments on different data sets. There are huge number of different variants of deep architectures, however, most of them are branched from some original parent architectures. Here I tried to introduce the mainstreams in this research area.

It is not always possible to compare the performance of multiple architectures all together since, they are not all implemented on the same data set. It is important to mention that deep learning is a fast growing field that one may find some different architectures, variants, or algorithms every couple of weeks. For instance I changed the contents of this chapter couple of times and add or remove some parts.

4.2.1 Deep Belief Networks

Deep Belief Networks (*DBNs*) are probably amongst the most famous and basic kind of deep neural network architectures. This is a generative probabilistic model with one visible layer at the bottom and many hidden layers upto the output. Each hidden layer unit learns the statistical representation via the links to the lower layers. The more higher the layers, the more complex are the representations [Lee et al., 2009a].

The main problem with the deep neural network architectures was the learning process, since the ordinary *gradient descent* algorithm does not work well and sometimes it makes the training quite impossible for a DBN. To circumvent this problem, a *greedy layerwise unsupervised pre-training* can be used [Hamel and Eck, 2010]. After the pre-training it is possible to do a *successful* supervised learning, done by a procedure called *fine-tuning*, using the renowned gradient descent. To paraphrase the problem, I shall say, the pre-training phase impacts on the choice of initial weights values for the actual supervised training stage. In practice it works magnificently better than the conventional random initialization of the weights, and causes to avoid *local minima* while using gradient descent in back propagation [Hamel and Eck, 2010; Plahl and Sainath, 2012].

DBNs are constructed by stacking many layers of restricted Boltzmann machines (RBMs), discussed in subsection 3.3.2,on top of each other [Hinton et al., 2006]. As depicted in Figure 4.2, an RBM comprises two layers, one is the visible and the other is hidden. Once we stack two RBMs on top of each other, the hidden layer of lower becomes visible to the top. The goal here is using the multiple layers of RBMs to model (represent) as close as possible to the reality, the distribution of the input. We are achieving this goal by multiple layers of non-linearity, which results in extraction the more accurate (more non-linear) probabilistic representation for the input [Hamel and Eck, 2010].

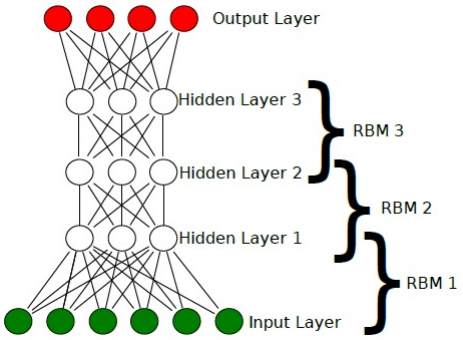


FIGURE 4.2: The schematic of a Deep Belief Network. The number of layer and the number of units on each layer in the schema are only examples [Hamel and Eck, 2010].

So far we have seen that the pre-training phase for the DBN is done through a greedy bottom-up pass. One may wonder about the possible information hidden in the reverse pass (top-down). This is one of the shortcoming of the so called pre-training for DBN-DNN architectures [Hinton et al., 2006]. Pre-training one layer of the network, while lower-layer weights remain unchanged causes the sub-optimality of the algorithm [You et al., 2013]. Recently, new learning algorithms for *deep Boltzmann machines* (*DBMs*) [Salakhutdinov and Hinton, 2009] were proposed. In the new architecture (DBM), a top-down pass is incorporated to the unsupervised generative pretraining process, so as to optimize the weights of different layers jointly together [Salakhutdinov and Hinton, 2009; Salakhutdinov and Larochelle, 2010]. DBMs can also be used for classification tasks by adding a top soft-max layer, see 4.2.2.

4.2.2 Deep Boltzmann Machines

A Deep Boltzmann Machine (DBM) is a type of binary pairwise Markov random field (undirected probabilistic graphical models) with multiple layers of hidden random variables. It is a network of symmetrically coupled stochastic binary units. It comprises a set of visible units $\boldsymbol{v} \in \{0,1\}^D$, and a series of layers of hidden units $\boldsymbol{h}^{(1)} \in \{0,1\}^{F_1}$,

$\boldsymbol{h}^{(2)} \in \{0,1\}^{F_2}, \cdots, \boldsymbol{h}^{(L)} \in \{0,1\}^{F_L}$. As it is depicted in Figure 4.3, there is no connection between the units of the same layer (like RBM). For the DBM of the Figure 4.3, we can write the probability which is assigned to vector $\boldsymbol{v}$ as:

$$p(\boldsymbol{v}) = \frac{1}{Z} \sum_{\boldsymbol{h}} e^{\sum_{ij} W_{ij}^{(1)} v_i h_j^1 + \sum_{jl} W_{jl}^{(2)} h_j^{(1)} h_l^{(2)} + \sum_{lm} W_{lm}^{(3)} h_l^{(2)} h_m^{(3)}} \tag{4.1}$$

where $\boldsymbol{h} = \{\boldsymbol{h}^{(1)}, \boldsymbol{h}^{(2)}, \boldsymbol{h}^{(3)}\}$ are the set of hidden units, and $\theta = \{\boldsymbol{W}^{(1)}, \boldsymbol{W}^{(2)}, \boldsymbol{W}^{(3)}\}$ are the model parameters, representing visible-hidden and hidden-hidden *symmetric interaction*, since they are undirected links, terms[1]. As it is clear by setting $\boldsymbol{W}^{(2)} = 0$ and $\boldsymbol{W}^{(3)} = 0$ the network becomes the well-known restricted Boltzmann machine, discussed in previous sections [Hinton and Salakhutdinov, 2012].

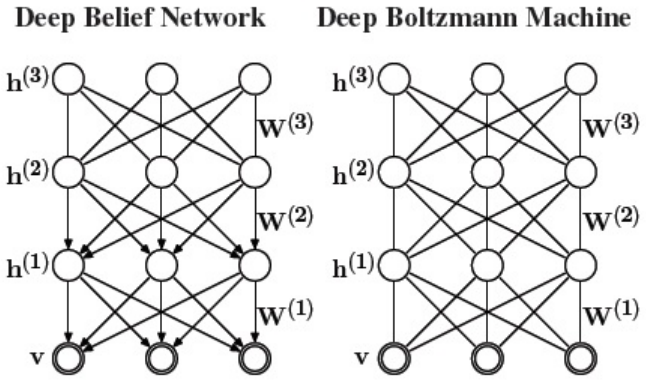


FIGURE 4.3: A deep belief network (left) and deep Boltzmann machine (right). The top two layers of a DBN form an undirected graph and the remaining layers form a belief net with directed, top-down connections. For a DBM, all the connections are undirected [Salakhutdinov and Larochelle, 2010].

There several reasons which motivate us to take advantage of deep Boltzmann machine architectures. Like DBNs they benefit from the ability of learning complex and abstract internal representations of the input in tasks such as object or speech recognition, with the use of *limited number* of *labeled* sensory data to fine-tune the representations which is built based on a *large supply* of *unlabeled* sensory input data. However, unlike DBNs and *deep convolutional* neural networks, they adopt the inference and training procedure in both directions, bottom-up and top-down pass, which enable the DBMs to better unveil the representations of the ambiguous and complex input structures [Salakhutdinov and Hinton, 2009; Bengio and LeCun, 2007]. Another important advantage of DBMs is the joint optimization of all layers using the approximate gradient of a variational lower-bound on the likelihood function which impacts greatly on the more proper learning of generative models [Salakhutdinov and Larochelle, 2010].

[1] We omit the bias terms for the clarity of the presentations.

Since the *exact maximum likelihood* learning is intractable for the DBMs, we may perform the *approximate maximum likelihood* learning. In order to do so, it is possible to use the learning procedure, discussed in subsection 3.3.1, for the general Boltzmann machines. However, one should note that this algorithm is rather slow, especially for those architectures with multiple layers of hidden units, where upper layers are quite remote from the visible layer [Salakhutdinov and Hinton, 2009]. There is another possibility, to use *mean-field* inference to estimate data-dependent expectations, incorporation with a *Markov Chain Monte Carlo* (*MCMC*) based stochastic approximation technique to approximate the expected *sufficient statistics* of the model [Hinton and Salakhutdinov, 2012]. Again, there is the problem of random initialization of weights, since the mentioned learning procedure performance is quite poor for DBMs. [Salakhutdinov and Hinton, 2009] introduced a layer-wise pre-training algorithm to solve or modify this problem.

In Figure 4.3, we can see the difference between DBN and DBM. In DBN the top two layers form a restricted Boltzmann machine which is an undirected graphical model, but the lower layers form a directed generative model. A greedy layer-wise unsupervised learning algorithm was introduced in [Hinton et al., 2006].

The idea behind the pre-training algorithm is straightforward. When learning parameters of the first layer RBM, the bottom-up weights are constrained to be twice the top-down weights and tie the visible-hidden weights, see Figure 4.4. Intuitively, using twice the weights when inferring the states of the hidden units $\boldsymbol{h}^{(1)}$ compensates for the initial lack of top-down feedback. Conversely, when pre-training the last RBM in the stack, the top-down weights are constrained to be twice the bottom-up weights. For all the intermediate RBMs the weights are halved in both directions when composing them to form a DBM, they are symmetric, as shown in Figure 4.4. This trick, eliminates the double-counting problem once top-down and bottom-up influences are subsequently combined. In this modified RBM with tied parameters, the conditional distributions over the hidden and visible states are defined as [Hinton and Salakhutdinov, 2012]:

$$p(\boldsymbol{v}) = \frac{1}{Z} \sum_{\boldsymbol{h}} e^{\sum_{ij} W_{ij}^{(1)} v_i h_j^1 + \sum_{jl} W_{jl}^{(2)} h_j^{(1)} h_l^{(2)} + \sum_{lm} W_{lm}^{(3)} h_l^{(2)} h_m^{(3)}} \tag{4.2}$$

This heuristic pre-training algorithm works surprisingly well in practice. However, it is solely motivated by the need to end up with a model that has symmetric weights, and does not provide any useful insights into what is happening during the pre-training stage. Furthermore, unlike the pre-training algorithm for Deep Belief Networks (DBNs), it lacks a proof that each time a layer is added to the DBM, the variational bound improves Hinton and Salakhutdinov [2012].

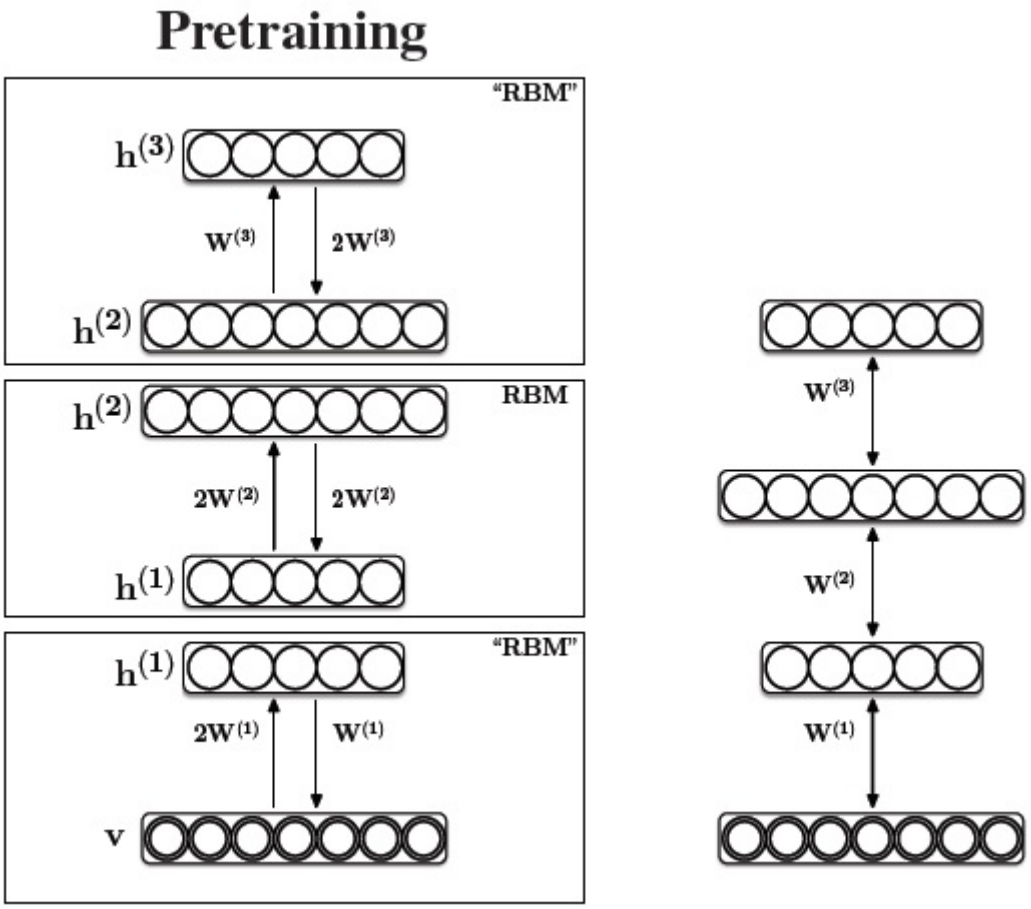


FIGURE 4.4: Pretraining a DBM with three hidden layers consists of learning a stack of RBMs that are then composed to create a DBM. The first and last RBMs in the stack need to be modified by using asymmetric weights [Hinton and Salakhutdinov, 2012].

Apart from all the advantages of DBMs discussed so far, they have a crucial disadvantage which limits the performance and functionality of this kind of architecture. The approximate inference, which is based on mean-field method, is about 25 to 50 times slower than a single bottom-up pass in DBNs. This time consuming task make the joint optimization, introduced before, quite impractical for large data sets, and also seriously restricts the use of DBMs in the tasks such as feature representations (the mean-field inference have to be performed for each new test input) [Salakhutdinov and Larochelle, 2010].

4.2.3 Deep Convolutional Neural Networks

Unlike DBMs, *Convolutional neural networks* are types of discriminative connectionist models. They are originally designed to operate directly on observed images without pre-processing [Arel et al., 2009a].

Once we are dealing with complex structures in data for tasks such as classification, we cannot use the simple multilayer perceptron, which has shown great performance for simple structures of data. This old method (MLP) has suffered from number of disadvantages due to factors such as the amount of training parameters, distortion (e.g., shifting or scaling), and ignoring the topology of the input. For instance, with a 24×24 input layer would already have 600 connections per single neuron in the hidden layer, which is really hard to train. To circumvent these problems, the process of classification has been divided into two major steps, feature extraction, done by a hand-crafted engineer with a priori knowledge about the problem at hand, and the classifier itself. The first step is a time consuming task which requires experts seeking for suitable features to extract [LeCun and Bottou, 1998].

Seeking for methods to tackle the problems of the MLPs, the basic convolutional neural networks was introduced in 1995 by Yann LeCun and Yoshua Bengio, two renowned computer scientists in the field [LeCun and Bengio, 1995]. It was inspired by the early work of Hubel and Wiesel in 1968 on cat's visual cortex [Hubel and Wiesel, 1968] and branched from MLP basic architecture. In [Hubel and Wiesel, 1968], it had been shown that cells are sensitive to small sub-regions of the input space, called a receptive field, and are tiled in such a way as to cover the entire visual field. One could think of them as local filters in input space and are thus better suited to exploit the strong spatially, local correlation presented in natural images [Hubel and Wiesel, 1968].

4.2.4 Stacked (Denoising) Auto-Encoders

In the context of neural network it is really common to believe that with several layers of non-linear processing one might be possible to model even a complex model efficiently, and to generalize the performance on difficult recognition tasks [McClelland et al., 1986; Hinton, 1989; Utgoff and Stracuzzi, 2002; Vincent et al., 2010]. This belief was inspired from both theoretical point of view such that in [Hastad, 1986; Hastad and Goldmann, 1991] and also from the discoveries related to the biological models of human brain such as visual cortex, for instance in [Bengio and LeCun, 2007; Bengio, 2009]. The non-convex characteristics of the optimization in MLPs had limited the scientists and engineers, for a long time, to apply more than two layers of hidden units [Bengio, 2009; Bengio et al., 2007]. Consequently, the researches had been carried out in *shallow* architectures so as to conceal the problem of optimization and cope with convex functions.

The autoencoder idea is motivated by the concept of *good* representation, in other words it states that not all the representations or features of the input are suitable to perform a specific task such as classification. Therefor, there is a need to have a clear understanding

of what is a *good* representation and how we can distinguish it. For instance for the case of classifier it is possible to define that *A good representation is one that will yield a better performing classifier* [Vincent et al., 2010]. Despite the fact that whether this philosophy might be true, we may think of it as a pre-training stage with a defined criterion.

According to the definition introduced in [Vincent et al., 2010], *encoder* is referred to a deterministic mapping f_θ that transforms an input vector $\boldsymbol{x}$ into hidden representation $\boldsymbol{y}$, where $\theta = \{\boldsymbol{W}, \boldsymbol{b}\}$, $\boldsymbol{W}$ is the weight matrix and $\boldsymbol{b}$ is an offset vector (bias). On the contrary a *decoder* maps back the hidden representation $\boldsymbol{y}$ to the reconstructed input $\boldsymbol{z}$ via $g_{\theta\prime}$. The whole process of autoencoding is to compare this reconstructed input to the original, apparently it needs an error criterion, and try to minimize this error to make the reconstructed value as close as possible to the original.

Here in the case of *denoising* autoencoders, we also focus on exploiting *good* representation. In this context we are seeking carefully the features which are useful for our specific task, and the rest (corrupting features) are not useful for the task at hand, so are to be *denoised* (*cleaned*). There are different strategies to distinguish and choose the *good* representations (features), such as restricting the representation by means of traditional *bottleneck* or *sparse* representation, maximizing the *mutual information*. However, in *stacked denoising autoencoders*, the partially corrupted output is cleaned (*denoised*). This fact has been introduced in [Vincent et al., 2010] with a specific approach to *good* representation, *a good representation is one that can be obtained robustly from a corrupted input and that will be useful for recovering the corresponding clean input*. Implicit in this definition are the ideas of *a*) The higher level representations are relatively stable and robust to the corruption of the input; *b*) It is required to extract features that are useful for representation of the input distribution .

The algorithm is performed by small changes in the basic autoencoders described above. It consists of multiple steps, starts by a stochastic mapping of $\boldsymbol{x}$ to $\tilde{\boldsymbol{x}}$ through $q_{\mathcal{D}}(\tilde{\boldsymbol{x}}|\boldsymbol{x})$, this is the corrupting step. Then the corrupted input $\tilde{\boldsymbol{x}}$ passes through a basic autoencoder process and is mapped to a hidden representation $\boldsymbol{y} = f_\theta(\tilde{\boldsymbol{x}}) = s(\boldsymbol{W}\tilde{\boldsymbol{x}} + \boldsymbol{b})$. From this hidden representation we can reconstruct $\boldsymbol{z} = g_{\theta\prime}(\boldsymbol{y})$. The whole process is depicted in Figure 4.5. In last stage a minimization algorithm is done in order to have a $\boldsymbol{z}$ as close as possible to uncorrupted input $\boldsymbol{x}$, the only difference with the conventional autoencoder is that $\boldsymbol{z}$ is adeterministic function of corrupted input $\tilde{\boldsymbol{x}}$ rather than uncorrupted input $\boldsymbol{x}$. The reconstruction error $L_H(\boldsymbol{x}, \boldsymbol{z})$ might be either the cross-entropy loss with an affine-sigmoid decoder, or the squared error loss with an affine decoder. Note that in this architecture parameters initialized randomly and adjusted using stochastic gradient descent algorithm [Vincent et al., 2010].

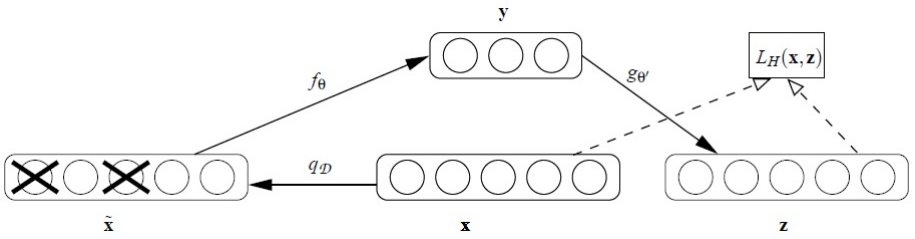


FIGURE 4.5: The denoising autoencoder architecture. An example $\boldsymbol{x}$, is stochastically corrupted, via $q_\mathcal{D}$, to $\tilde{\boldsymbol{x}}$. Then autoencoder then maps it to $\boldsymbol{y}$ via encoder f_θ and attempts to reconstruct $\boldsymbol{x}$ via decoder $g_{\theta\prime}$, producing reconstruction $\boldsymbol{z}$. Reconstruction error is measured by loss $L_H(\boldsymbol{x}, \boldsymbol{z})$ [Vincent et al., 2010].

In order to make a deep architecture out of this denoising autoencoders, we have to stack them one on top of another, similar to what I have already mentioned for the RBMs in DBNs or what is done in [Bengio et al., 2007; Ranzato et al., 2006; Larochelle et al., 2009] for conventional autoencoders. Once the encoding function f_θ of the first denoising autoencoder is learned and used to uncorrupt the input (corrupted input), we can train the second level of stacked autoencoder, the complete process is shown in Figure 4.6 [Vincent et al., 2010].

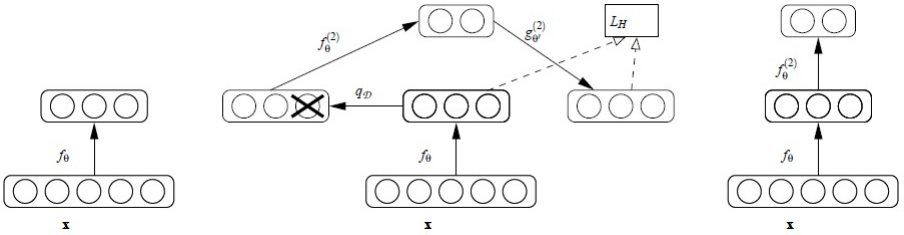


FIGURE 4.6: Stacking denoising autoencoder. After training a first level denoising autoencoder, see Figure 4.5, its learned encoding function f_θ is used on clean input (left). The resulting representation is used to train a second level denoising autoencoder (middle) to learn a second level encoding function $f_\theta^{(2)}$. From there, the procedure can be repeated (right) [Vincent et al., 2010].

Once the stacked autowncoder is trained, its output might be used as the input to a supervised learning algorithm such as *support vector machine* classifier or a multi-class logistic regression. For a complete discussion and more details such as geometrical interpretation, types of corruption and related approaches I refer the interested reader to [Vincent et al., 2010].

4.2.5 Deep Stacking Networks

One of the deep architectures recently introduced in [Deng and Yu, 2011a,b], which is based on building hierarchies with blocks of simplified neural network modules, is called *deep convex network*. They are called convex because of the formulation of the weights

learning problem, which is convex optimization problem with a closed-form solution while the lower layer weights are initialized with a fixed RBM. However the network was renamed the *deep stacking network* (*DSN*) [Deng et al., 2012], emphasizing on this fact that a similar mechanism as the *stacked generalization* is used [Wolpert, 1992]. Along with this new name, there is a change in the learning process, for better performance in tasks such as classification, the lower-layer weights are also learned, which causes the overall learning problem is not a convex problem any longer [Hutchinson et al., 2012a].

The DSN blocks, each consisting of a simple, easy-to-learn module, are stacked to form the overall deep network. It can be trained block-wise in a supervised fashion without the need for back-propagation for the entire blocks [Bengio, 2009].

As designed in [Deng and Yu, 2011a,b] each block consists of a simplified MLP with a single hidden layer see Figure 4.7. It comprises a weight matrix $\boldsymbol{U}$ as the connection between the logistic sigmoidal units of the hidden layer $\boldsymbol{h}$ to the linear output layer $\boldsymbol{y}$, and a weight matrix $\boldsymbol{W}$ which connects each input of the blocks to their respective hidden layers. If we assume that the target vectors $\boldsymbol{t}$ be arranged to form the columns of $\boldsymbol{T}$ (the target matrix), let the input data vectors $\boldsymbol{x}$ be arranged to form the columns of $\boldsymbol{X}$, let $\boldsymbol{H} = \sigma(\boldsymbol{W}^T\boldsymbol{X})$ denote the matrix of hidden units, and assume the lower-layer weights $\boldsymbol{W}$ are known (training layer-by-layer). The function σ performs the element-wise logistic sigmoid operation. Then learning the upper-layer weight matrix $\boldsymbol{U}$ can be formulated as a convex optimization problem:

$$\underset{\boldsymbol{U}^T}{\text{minimize}} \quad f = \| \boldsymbol{U}^T\boldsymbol{H} - \boldsymbol{T} \|_F^2 \tag{4.3}$$

which has a closed-form solution. The input to the first block $\boldsymbol{X}$ only contains the original data, however in the upper blocks in addition to this original (raw) data there is a copy of the lower-block(s) output y, see Figure 4.7. It is possible to optimize the weights of the layer-layer (input-hidden links) using an accelerated gradient descent [Yu and Deng, 2011], this is done by minimizing the squared error objective in 4.3. In this step we assume that upper-layers weights $\boldsymbol{U}$ are known (the optimal solution from the previous step) and try to update the $\boldsymbol{W}$ using iterative gradient algorithm, then again $\boldsymbol{U}$ shall be computed with new values [Hutchinson et al., 2012a].

What is done in this method is in each block an estimate of the same final label class $\boldsymbol{y}$ is produced, then this estimated label concatenated with original input to form the *expanded input* for the upper block. Note that here in this architecture in contrast with other deep architectures, such as DBNs, the goal is not to discover the transformed feature representation. Regarding the structure of the hierarchy of this kind of architecture, it makes the parallel training possible. I have to mention that in purely discriminative

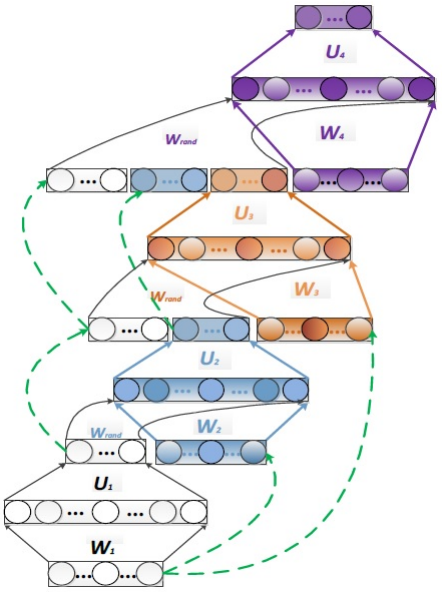


FIGURE 4.7: A DSN architecture. Only four modules are illustrated, each with a distinct color. Dashed lines denote copying layers [Deng, 2013].

tasks DSN performance is better than the conventional DBN [Deng et al., 2012]. Note that only in the final block the output $\boldsymbol{y}$ is used for the classification tasks, in other blocks the output is only used to form the expanded input for the upper blocks

4.2.6 Tensor Deep Stacking Networks (T-DSN)

The architecture discussed here is an extension of the DSN introduced in previous subsection 4.2.5. It is called *tensor deep stacking network* (*TDSN*). It improves the DSN in two important ways, using the higher order information by means of covariance statistics and also transforming the non-convex problem of the lower-layer to a convex sub-problem of the upper-layer [Hutchinson et al., 2012a].

Unlike the DSN, the covariance statistics of the data is employed using a bilinear mapping from two distinct set of hidden units in the same layer, Figure 4.8, to predictions via a third-order tensor. Looking to the covariance structure of the data was also proposed in the works on *mean-covariance RBM* (*mcRBM*) architecture, however, there they use it on the raw data rather than on binary hidden feature layers as is done in TDSN [Dahl et al., 2010; Ranzato et al., 2010a; Ranzato and Hinton, 2010]. In the mcRBM the higher-order structure is represented in the visible data, while in the TDSN it is the hidden units which are responsible for this representation. Due to the learning complexity of the mcRBM models which are caused by the factorization, required for the reduction of the cubic growth in the size of the weight parameters, it is also very difficult to use mcRBM in deeper layers, usually it is used only in the bottom layer of a

deep architecture. These difficulties of the factorization in addition to the high costs of the Hybrid Monte Carlo in learning are the very limiting factors in mcRBM to scale up to very large data sets. However, they are all removed in TDSN, and due to the specific architecture of TDSN the parallel training and closed-form solution for the upper-layer convex problems are straightforward. TDSN adopts small sizes of hidden layers so as to eliminate the factorization process. There are other differences between mcRBM and TDSN. The mcRBM is a is a generative model optimizing a maximum likelihood objective, while the TDSN is a discriminative model optimizing a least squares objective. In [Hutchinson et al., 2012b], more advantages of the TDSN over other architectures are discussed.

The scalability and parallelization are the two important factors in the learning algorithms which are not considered seriously in the conventional DNNs [Hinton and Salakhutdinov, 2006; Dahl et al., 2012; Mohamed et al., 2012]. In [Deng and Yu, 2011a,b; Deng et al., 2012], is noted that all the learning process for the DSN (and TDSN as well) is done on a batch-mode basis, so as to make the parallelization possible on a cluster of CPU and/or GPU nodes. Parallelization gives us the opportunity to scale up our design to larger (deeper) architectures and data sets, in a way different than what is done in [Le et al., 2011] for deep sparse autoencoders.

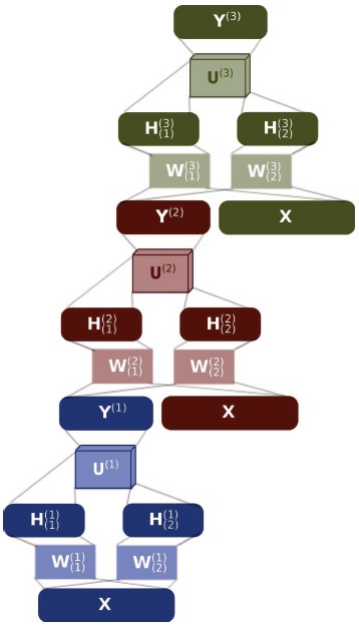


FIGURE 4.8: An example TDSN architecture with three stacking blocks, where each block consists of three layers, and superscript is used to indicate the block number. Inputs ($\boldsymbol{X}$) and outputs ($\boldsymbol{Y}^{(i-1)}$) are concatenated to link two adjacent blocks. The hidden layer in each block has two parallel branches ($\boldsymbol{H}^{(i)}_{(1)}$ and $\boldsymbol{H}^{(i)}_{(2)}$) Hutchinson et al. [2012a].

In Figure 4.8 a modular representation of the TDSN architecture is depicted. Three blocks, each of which has two sets of lower-layer (input-hidden) weights $\boldsymbol{W}_{(1)}$ and $\boldsymbol{W}_{(2)}$, are stacked on top of each other to form a TDSN architecture. Unlike DSN, each input ($\boldsymbol{X}$ and a copy of the output of previous layers if there is any) layer is connected to two parallel hidden layers (in the same layer) $\boldsymbol{H}_{(1)}$ and $\boldsymbol{H}_{(2)}$. There is also a three-way (connecting the two parallel hidden layer to the output layer) upper-layer weight tensor (not matrix) $\mathcal{U}$.

It is important to note that the basic architecture shown in Figure 4.8 is suitable for tasks such as classification, or regression. However, if it is used as a part of a hybrid architecture with HMM, a softmax layer to produce posterior probabilities is desirable [Hutchinson et al., 2012a].

4.2.7 Spike-and-Slab RBMs (ssRBMs)

The need for real-valued inputs which are employed in Gaussian RBMs (GRBMs), motivates scientists seeking new methods. One of these methods is the *spike and slab RBM* (*ssRBM*), which models continuous-valued inputs with strictly binary latent variables Courville et al. [2011a].

Similar to basic RBMs and its variants, the spike and slab RBM is a bipartite graph. Like GRBM the visible units (input) are real-valued. The difference arises in the hidden layer, where each hidden unit come along with a binary spike variable and real-valued slab variable. These terms (spike and slab) come from the statistics literature [Mitchell and Beauchamp, 1988], and refer to a prior including a mixture of two components. One is a discrete probability mass at zero called spike, and the other is a density over continuous domain [Courville et al., 2011b].

There is also an extension of the ssRBM model, which is called μ-ssRBM. This variant provides extra modeling capacity to the architecture using additional terms in the energy function. One of these terms enable model to form a conditional distribution of the spike variables by means of marginalizing out the slab variables given an observation, which is quite similar to the conditional of mcRBM method [Ranzato and Hinton, 2010] and also mPoT model [Ranzato et al., 2010b]. The μ-ssRBM slab variables and input are jointly Gaussian with diagonal covariance matrix, given both observed and spike variables. The observations are Gaussian with diagonal covariance, given both the spike and slab variables. The μ-ssRBM is related to Gibbs sampling. These properties make this model as a good choice for the building blocks of deep structures such as DBM. However, there

is no guarantee that the resulting model produce a valid density over the whole real-valued data space, this is one of the main shortcomings of ssRBM architectures [Courville et al., 2011b].

4.2.8 Compound Hierarchical-Deep Models

For a human brain in comparison with the current state-of-the-art artificial systems, fewer number of examples is needed to categorize and even extend the already existing categories for the novel instances (generalization) [Kemp et al., 2007; Perfors and Tenenbaum, 2009; Smith et al., 2002; Xu and Tenenbaum, 2007]. This is the main motivation of this subsection, by means of learning abstract knowledge of the data and use them for novel cases in the future [Salakhutdinov et al., 2012].

The class of these new architectures is called *compound HD models*, where *HD* stands for *Hierarchical-Deep*. They are structured as a composition of non-parametric Bayesian models with deep networks. The features, learned by deep architectures such as DBNs [Hinton et al., 2006], DBMs [Salakhutdinov and Hinton, 2009], deep autoencoders [Larochelle et al., 2009], convolutional variants [Coates et al., 2011; Lee et al., 2009b], ssRBMs [Courville et al., 2011b], deep coding network [Lin et al., 2010], DBNs with sparse feature learning [Ranzato et al., 2007], recursive neural networks [Socher et al., 2011], conditional DBNs [Taylor et al., 2006], denoising autoencoders [Vincent et al., 2008], are able to provide better representation for more rapid and accurate classification tasks with high-dimensional training data sets. However, they are not quite powerful in learning novel classes with few examples, themselves. In these architectures, all units through the network are involved in the representation of the input (*distributed representations*), and they have to be adjusted together (high degree of freedom). However, if we limit the degree of freedom, we make it easier for the model to learn new classes out of few training samples (less parameters to learn). *Hierarchical Bayesian* (*HB*) models, provide us learning from few examples as you may find in [Kemp et al., 2007; Xu and Tenenbaum, 2007; Chen et al., 2011; Fei-Fei et al., 2006; Rodriguez et al., 2008] for computer vision,, statistics, and cognitive science. These HB models are based on categorization of the training examples, and the generalization to the new classes at hands. However, they are all fed by hand-crafted features [Fei-Fei et al., 2006; Bart et al., 2008] such as GIST, or SIFT features in computer vision, and MFCC features in speech perception domains. Another shortcoming of HB models is the fixed architecture it employs [Canini and Griffiths, 2009; Sudderth et al., 2008], it does not discover the representation and the links between different parameters in an unsupervised fashion [Salakhutdinov et al., 2012].

There are several methods addressing the subject of learning with few examples, such as using several boosted detectors in a multi-task settings in [Torralba et al., 2006], cross-generalization approach in [Bart and Ullman], which are both discriminative, boosting methods in [Babenko et al., 2009], and also HB approach in [Fei-Fei et al., 2006].

Compound HD architectures try to integrate both characteristics of HB and deep networks. Here we introduce the compound HDP-DBM architecture, a *hierarchical Dirichlet process* (*HDP*) as a hierarchical model, incorporated with DBM architecture. It is a full generative model, generalized from abstract concepts flowing through the layers of the model, which is able to synthesize new examples in novel classes that look *reasonably natural*. Note that all the levels are learned jointly by maximizing a joint log-probability score [Salakhutdinov et al., 2012].

Consider a DBM with three hidden layers, the probability of a visible input $\boldsymbol{v}$ is:

$$P(\boldsymbol{v};\psi) = \frac{1}{\mathcal{Z}} \sum_{\boldsymbol{h}} exp\left(\sum_{ij} \boldsymbol{W}_{ij}^{(1)} v_i h_j^1 + \sum_{jl} \boldsymbol{W}_{jl}^{(2)} h_j^1 h_l^2 + \sum_{lm} \boldsymbol{W}_{lm}^{(2)} h_l^2 h_m^3 \right) \tag{4.4}$$

where $\boldsymbol{h} = \{\boldsymbol{h}^1, \boldsymbol{h}^2, \boldsymbol{h}^3\}$ are the set of hidden units, and $\psi = \{\boldsymbol{W}^{(1)}, \boldsymbol{W}^{(2)}, \boldsymbol{W}^{(3)}\}$ are the model parameters, representing visible-hidden and hidden-hidden symmetric interaction terms.

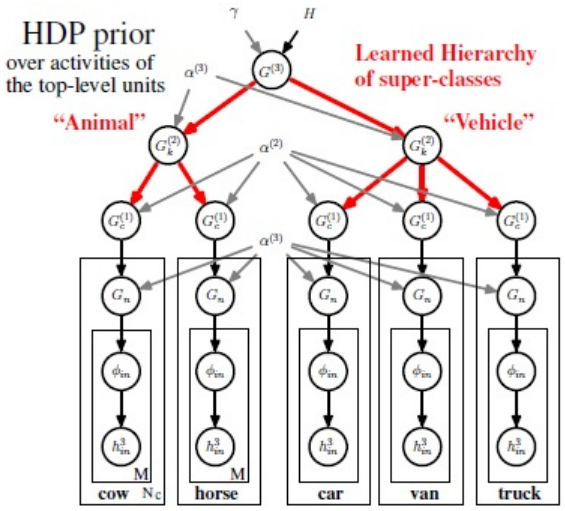


FIGURE 4.9: Hierarchical Dirichlet Process prior over the states of $\mathbf{h}^3$ [Salakhutdinov et al., 2011].

After a DBM model has been learned, we have an undirected model that defines the joint distribution $P(\boldsymbol{v}, \boldsymbol{h}^1, \boldsymbol{h}^2, \boldsymbol{h}^3)$. One way to express what has been learned is the conditional model $P(\boldsymbol{v}, \boldsymbol{h}^1, \boldsymbol{h}^2 | \boldsymbol{h}^3)$ and a prior term $P(\boldsymbol{h}^3)$. We can therefore rewrite the variational bound as:

$$logP(\boldsymbol{v}) \geq \sum_{\boldsymbol{h}^1,\boldsymbol{h}^2,\boldsymbol{h}^3} Q(\boldsymbol{h}|\boldsymbol{v};\mu) logP(\boldsymbol{v},\boldsymbol{h}^1,\boldsymbol{h}^2|\boldsymbol{h}^3) + \mathcal{H}(Q) + \sum_{\boldsymbol{h}^3} Q(\boldsymbol{h}^3|\boldsymbol{v};\mu) logP(\boldsymbol{h}^3) \quad (4.5)$$

This particular decomposition lies at the core of the greedy recursive pre-training algorithm: we keep the learned conditional model $P(\boldsymbol{v},\boldsymbol{h}^1,\boldsymbol{h}^2|\boldsymbol{h}^3)$, but maximize the variational lower-bound of 4.5 with respect to the last term [Hinton et al., 2006]. Instead of adding an additional undirected layer, (e.g. a restricted Boltzmann machine), to model $P(\boldsymbol{h}^3)$, we can place a hierarchical Dirichlet process prior over $\boldsymbol{h}^3$, that will allow us to learn category hierarchies, and more importantly, useful representations of classes that contain few training examples. The part we keep, $P(\boldsymbol{v},\boldsymbol{h}^1,\boldsymbol{h}^2|\boldsymbol{h}^3)$, represents a *conditional* DBM model, which can be viewed as a two-layer DBM but with bias terms given by the states of $\boldsymbol{h}^3$:

$$P(\boldsymbol{v},\boldsymbol{h}^1,\boldsymbol{h}^2|\boldsymbol{h}^3) = \frac{1}{\mathcal{Z}(\psi,\boldsymbol{h}^3)} exp(\sum_{ij} \boldsymbol{W}^{(1)}_{ij} v_i h^1_j \sum_{jl} \boldsymbol{W}^{(2)}_{jl} h^1_j h^2_l \sum_{lm} \boldsymbol{W}^{(3)}_{lm} h^2_l h^3_m) \quad (4.6)$$

4.2.9 Convolutional Deep Belief Networks

Convolutional DBNs are used for high dimensional data, they are trained in a greedy, bottom-up fashion similar to DBNs. They have shown great success [Lee et al., 2009b] in visual recognition tasks [Lee et al., 2009a]. They are also applied to some unlabeled audio data sets such as speech and music, and their feature representation outperform other baseline features such as spectrogram and MFCC for multiple audio classification tasks [Lee et al., 2009a].

As RBMs are the building blocks for DBNs, Convolutional DBNs comprise multiple convolutional RBMs [Lee et al., 2009b; Norouzi et al., 2009; Desjardins and Bengio, 2008]. In this new convolutional settings the visible-hidden weights are shared among all locations in the hidden layer. For convolutional RBMs, visible layer V might be binary or real-valued however the hidden layer H is binary. We can perform *block Gibbs sampling* inference due to the conditional independence between the units in one layer given the units of other layer. A *probabilistic max-pooling* is also developed for convolutional RBMs in [Lee et al., 2009b] and [Lee et al., 2009a] use this method for convolutional DBNs. They did not employ the exact maximum likelihood, instead implemented with contrastive divergence mentioned in [Hinton, 2002] to approximate the gradients. Usually a sparsity penalty term is added to the log-likelihood objective [Lee et al., 2009b, 2007]. It might be viewed as restricting the *capacity* which is often results

in more easily interpretable feature representations. Whenever, each convolutional RBM is trained, we shall stack them on top of each other to build a convolutional DBN. It is possible to use feed-forward approximation as inference method [Lee et al., 2009a].

4.2.10 Deep Coding Networks

There are several advantages to have a model which can *actively* update itself to the context in data. One of these methods arises from this idea to have a model which is able to adjust its prior knowledge dynamically according to the context of the data. *Deep coding network* (*DPCN*) is a predictive coding scheme where top-down information are used to empirically adjust the priors needed for the bottom-up inference procedure by means of a deep locally-connected generative model. This is based on extracting sparse features out of time-varying observations using a linear dynamical model. Then with an extension to this feature extraction block, a pooling strategy is employed in order to learn invariant feature representations. Similar to other deep architectures, these blocks are the building elements of a deeper architecture where greedy layer-wise unsupervised learning are used. Note that the layers constitute a kind of Markov chain such that the states at any layer are only dependent on the succeeding and preceding layers. Since, it predicts the representation of the layer, by means of a top-down approach using the information in upper layer and also temporal dependencies from the previous states, it is called deep predictive coding network (DPCN) [Chalasani and Principe, 2013].

In [Rao and Ballard, 1997; Friston, 2008], a statistical model is also used to explain the cortical functions in the mammalian brain. Those model are in a close relation with the one introduced here, DPCN. [Rao and Ballard, 1997] uses a kind of update procedure like Kalman filter for inference and a general framework consisting of all higher-order moments is used in [Friston, 2008]. The important problem with these methods is the lack of discriminative representation (sparse and invariant representation), helpful for task such as object recognition. However, in DPCN, and efficient inference algorithm is employed to extract locally invariant representation of the image sequences and more abstract information in higher layers. It is also possible to extend the DPCN to from a convolutional network [Chalasani and Principe, 2013].

4.2.11 Deep Kernel Machines

As I mentioned before artificial neural network is not the only area conquered by deep concept. The *Multilayer Kernel Machine* (*MKM*) as introduced in [Cho, 2012] is one of those deep architectures which are not in the field of neural network, albeit, quite relevant. It is a way of learning highly nonlinear functions with the iterative applications

of weakly nonlinear kernels. They use the *kernel principle component analysis* (*KPCA*), in [Schölkopf et al., 1998], as method for unsupervised greedy layer-wise pre-training step of the deep learning architecture.

In this method, layer $\ell + 1^{th}$ learns the representation of the previous layer ℓ, extracting the n_ℓ principle component (PC) of the projection layer ℓ output in the feature domain induced by the kernel. For the sake of dimensionality reduction of the updated representation in each layer, a supervised strategy is proposed to select the best informative features among the ones extracted by KPCA. We can numerate this process as follows:

1. ranking the n_ℓ features according to their mutual information with the class labels.
2. for different values of K and $m_\ell \in \{1, \ldots, n_\ell\}$, compute the classification error rate of a *K-nearest neighbor* (*K-NN*) classifier using only the m_ℓ most informative features on a validation set.
3. the value of m_ℓ with which the classifier has reached the lowest error rate determines the number of features to retain.

There are some drawbacks in using the KPCA method as the building cells of an MKM. It is a time consuming task, as the cross validation stage of the feature selection process needs quite long time. To circumvent this shortcoming, the use of a more efficient kernel method is used in [Yger et al., 2011], called the *kernel partial least squares* (*KPLS*). This new approach remove the cross validation stage, and merge the selection process into the projection strategy [Rosipal et al., 2003]. The features are selected in an iterative supervised fashion, where the KPLS selects the j^{th} feature that most correlated with the class labels, solving an updated eigen-problem, at each iteration j. In this method, the eigenvalue λ_j of the extracted feature indicates the discriminative importance of this feature. the number of iterations to be done is equal to the number of features to extract by KPLS, and is determined by a thresholding of λ_j [Cho, 2012].

Chapter 5

Deep Architectures for Sequential Patterns

Representation learning make the systems to be able to rely more on the information extracted from the data itself, applying some processing stages, rather than the prior knowledge about a specific task given from an outer source [Hamel et al., 2011]. regarding this principle, we have studied several methods to better extract and deploy this knowledge, potentially outsourced by the original data. In this chapter we will focus on architectures which are concerned with the sequential data, such as speech or motion.

5.1 DBN-HMM (Hybrid Approach)

The *hybrid models*, which is predicated on the methods employ both neural network and HMM for the case of sequential patterns such as what has been proposed in [Bourlard and Morgan, 1994; Dahl et al., 2012; Mohamed et al., 2012; Renals and Morgan, 1994; Mohamed et al., 2010; Yu et al., 2010a; Seide et al., 2011a,b; Yu et al., 2012a; Hinton et al., 2012] for speech recognition. This old method is recently resurrected due to the advances occurred in the area of *high-speed general purpose graphical processing units* (*GPGPUs*) and also new explorations in deep neural network architectures and algorithms [Yu et al., 2012b]. For instance in speech, with the new approaches and methods introduced in different papers and research groups, it has shown that many deep neural network architectures are able to outperform the conventional GMMs at acoustic modeling in a variety of data sets [Hinton et al., 2012].

There are different architectures of hybrid models either with shallow or deep structure, however there is something in common between all these variety of methods, the

combination of neural networks and HMMs. In this section we will discuss about the DBN-HMM. Although, there are different variants of DBN-HMM recently proposed, we will talk about the main concepts and principles. In principle, neural networks are used to extract the probabilities needed for *acoustic modeling*. They play the role of temporal ordering, which is a static pattern, while HMMs used to model the dynamic properties of the data. To the best of my knowledge the state-of-the-art in this field is the method proposed in [Dahl et al., 2010], which itself a modification of the previous works introduced in [Mohamed et al., 2012, 2009].

Using the method in [Mohamed et al., 2009], DBNs are trained with an unsupervised fashion as a pre-training stage, introduce in [Hinton et al., 2006], followed by a supervised fine-tuning such as back-propagation algorithm, in order to model posterior densities over HMM states for a given frame of acoustic input. In this method, n adjacent frames of acoustic input (each frame as a vector of acoustic features) fed into the DBNs as the training set. In the supervised learning phase, the cross-entropy loss for each HMM state predictions, is optimized. In the conventional approaches, usually GMM is used instead of DBN. As a comparison with the traditional method, in [Mohamed et al., 2009] the posterior distribution over HMM states is used.

5.2 Conditional DBNs

The building blocks of a *Conditional Deep Belief Network* are *Conditional* Restricted Boltzmann Machines (CRBMs). This conditional dependency deals with the temporal information contents in data. Ordinary RBMs do not convey this kind of information. It is possible to treat the previous time slice(s) as additional fixed input to the network. As it is illustrated in Figure 5.1, there are two kinds of connections in the architecture; Connections from the past visible units N to the current visible units (interlayer connections), and connections from past visible units M to the current hidden units (intralayer connections). Since it is a conditional dependency, all these links are directed (conveying arrows). This new configuration is called a conditional RBM. Now we may say there are two dimensions, one is horizontal and directed (depth in time) and the other is vertical and undirected [Taylor, 2009].

N and M are the two important parameters to show the depth of time dependency. Unlike LSTM (see section 5.7), in this method there is no automatic approach for choosing these two parameters, and need to be fixed by hand regarding the task at hand. For instance in modeling the motion they are set such that $N = F/10$, $N = M$, where F is the frame rate of the data.

If, for simplifying our discussion, we assume that $N = M$, and also consider $\{t-1, \ldots, t-N\}$ as the N previous time steps, then we can concatenate all the N previous time slices of input in a vector called *history* vector $v_{<t}$. So if each input is of dimension D then the dimension of the history vector $v_{<t}$ is of dimension $N \cdot D$. There are also two new weight matrices, one for the past visible to current which is of dimension $N \cdot D \times D$ called $A$and the other related to the past visible to current hidden with the dimension of $N \cdot D \times H$ called B where the H is the number binary hidden units [Taylor, 2009].

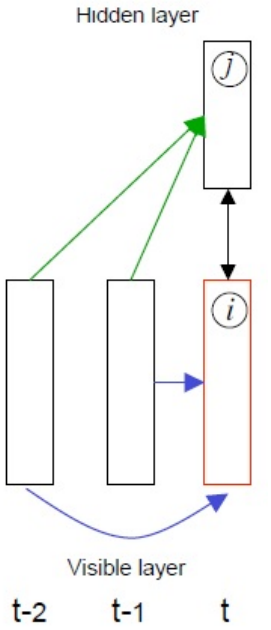


FIGURE 5.1: Architecture of a Conditional Restricted Boltzmann Machine. In this figure we show $N = 2$ but in the experiments, it is typically of a slightly higher order [Taylor, 2009].

It is useful to think of each past input to the current hidden layer as a dynamic bias term. The inference and training algorithm in CRBMs is not more difficult than the standard RBMs. It is still possible to use the contrastive divergence for the training. Like the ordinary DBN, once the CRBMs are trained, we can stack them on top of each other to form a conditional DBN, the hidden state vectors of the lower level are considered as a *fully observed* data for the upper level [Taylor, 2009].

As it is depicted in Figure 5.2, (a) shows a CRBM which is at level zero (the bottom level). W_0 is the top-down generative weights and W_0^T is the bottom-up recognition weight matrix. The diagram is completed at (d), what we have introduce in the previous paragraph forming a conditional DBN by stacking the CRBMs. For the joint connection (undirected) of visible-hidden connection in a CRBM, a joint distribution is defined as $p(v_t^0, h_t^0 | v_{<t}^0)$, which could be decomposed to a multiplication of $p(v_t^0 | h_t^0, v_{<t}^0)$ (mapping from features to data) and $p(h_t^0 | v_{<t}^0)$ (prior over features), according to the product rule of probability. For a more complete discussion of the algorithms, one can refer to

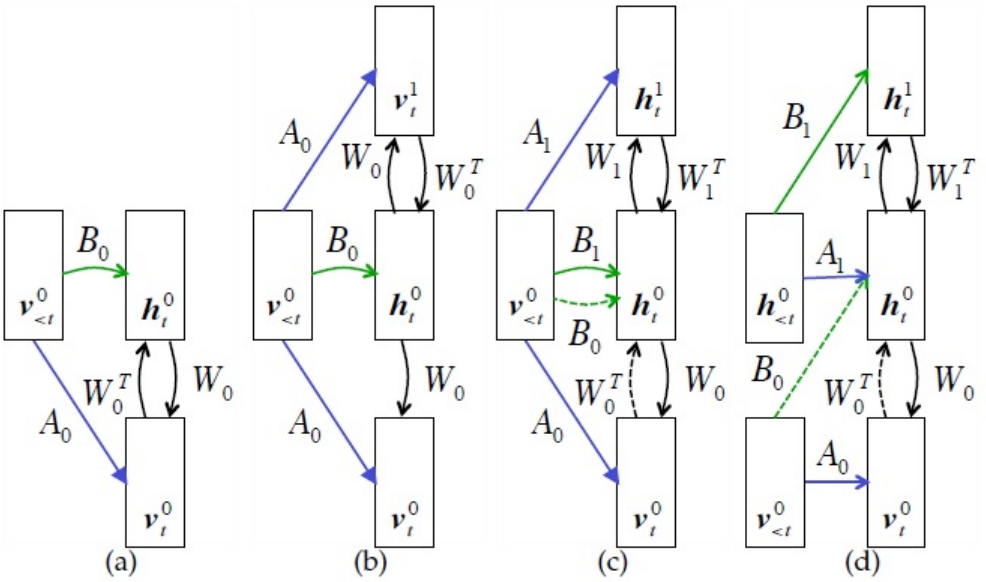


FIGURE 5.2: Building a Conditional Deep Belief Network. (a) The CRBM. (b) A generative model whose weights between layers are tied. It defines the same joint distribution over v_t^0 and h_t^0. The top two layers interact using symmetric connections while all other connections are directed. (c) Improving the model by untying the weights; holding W_0, A_0 and B_0 fixed and greedily training W_1, A_1 and B_1. Note that the *dashed* directed, bottom-up weights are not part of the generative model. They are used to infer factorial, approximate posterior distributions over h_t^0 when v_t^0 is clamped to the data. (d) The model we use in practice. We ignore uncertainty in the past hidden states [Taylor, 2009].

[Taylor, 2009], there you may find the *factored CRBM* an interesting topic and also a comparison with HMM approach.

5.3 Temporal Restricted Boltzmann Machine

There are different architectures and variants known as *temporal RBM* (*TRBM*) such as the ones proposed in [Garg and Henderson, 2011; Sutskever and Hinton, 2007]. In this section we will deal with the one in [Garg and Henderson, 2011]. TRBMs are possible to exploit information about the past time steps (horizontal depth) [Taylor et al., 2006]. As you can see in Figure 5.3, TRBM is very similar to the conditional RBM, not quite the same, however. In a TRBM, unlike the conditional RBM, there is no direct link between previous time steps and the current hidden layer. In a TRBM architectures it is possible to introduce time dependencies of more than one time step behind, which is not common in CRBMs.

Like many other deep architectures, a contrastive divergence algorithm is used for the training with only *one step reconstruction* [Hinton, 2002]. Since, there are limited number of visible values, it is possible to modify the approximation by taking the derivatives.

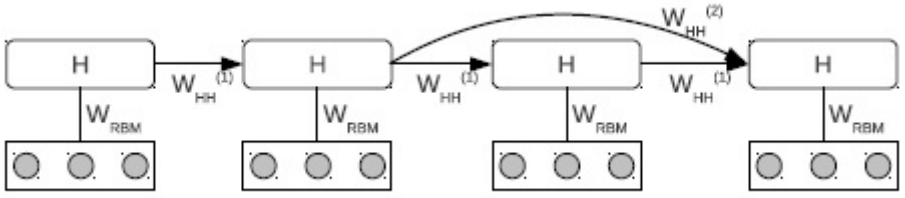


FIGURE 5.3: Temporal RBM Model. Edges with no arrows represent undirected RBM connections. The directed temporal connections between time steps contribute a bias to the latent layer inference in the current step [Garg and Henderson, 2011].

Using the back-propagation algorithm, the weights on the time dependency edges are learned. Note that similar to the Recurrent TRBM architecture of [Sutskever et al., 2008], it is possible to use future time steps to train the current RBM weights, however, in [Garg and Henderson, 2011] CD is not used for each individual time step, they use the beam-search as in [Titov and Henderson, 2007], instead.

5.4 Deep Spatio-Temporal Inference Network (DeSTIN)

The architecture, here is named *deep spatio-temporal inference network* (*DeSTIN*) was first inspired by the findings in neuroscientists' research on neocortex paradigm. It is thought that neocortex consists of many building blocks which are similar in essence, called cortical circuits and form a hierarchical architecture. This is the structure enables mammals to represent spatio-temporal dependencies in sensory information. The main idea is to divide high-dimensional sensory data into small segments and model these smaller parts [Arel et al., 2009a,b].

There is no explicit specific preprocessing in the neocortex, just propagating through a complex architecture and try to learn the abstract representation hidden in the pattern ready for the tasks such as classification [Barlow, 1989]. This is a discriminative learning architecture, which is a combination of unsupervised learning for dynamic patterns and Bayesian inference.

The architecture comprises multiple blocks, say nodes (cortical circuits), each of which characterizes the sequence of patterns from previous nodes. At the lowest layer (input layer), the raw data such as pixels of an image, is input to the system. In upper layers they receive a belief state of the preceding lower layers. The belief state is a probability mass function over the sequences of input, which is learned by the nodes. Therefore a predefined number of state variables is allotted to each node. There are two important abilities concerning the DeSTIN. In DeSTIN both spatial and temporal dependencies in data are captured, forming the belief space through the layers. Each processing node

is identical, which results in ease of using parallel processing [Arel et al., 2009b]. For a quantitative analysis of the architecture I refer the interested reader to [Karnowski, 2012].

In order to show the significance of DeSTIN architecture, it is possible to make a comparison between this architecture and other currently used schematics with the deep concept. It is possible to modify the DeSTIN architecture to benefit from other advantages of different DML methods. The most significant difference between DeSTIN and conventional CNNs or DBNs is the inherent temporal representation ability of the DeSTIN architecture. The CNNs are image-based which can represent spatial dependencies of the data, however, it is possible to be extended for temporal representations with an expensive network complexity and also fix estimate of the temporal window size. In contrast, DeSTIN works without any prior estimation of the temporal window size, but rather learns it adaptively.

Another important attribute of DeSTIN is distinguished when you compare it with the work done in [Mobahi et al., 2009]. In the so called paper, the concept of temporal coherence (adjacent video frames contain the same object) is used. With this concept a CNN is trained so as to make the adjacent time steps outputs as similar or close as possible to each other, by means of some modified version of back-propagation algorithm. It employs two different cost functions one for supervised learning, which is an ordinary gradient descent algorithm, and the other is an unspervised learning algorithm. It actually incorporates temporal knowledge into a semi-supervised approach. However, in the DeSTIN, they use a single learning algorithm. Extensions to the RBM to build DBNs to include temporal information are mentioned in section 5.3 and also in [Sutskever and Hinton, 2007; Lockett and Miikkulainen, 2009]. However, these extensions suffer from the expensive training time, and also the size of the time window must be predefined explicitly [Karnowski, 2012].

Note that the DeSTIN is inherently an unsupervised algorithm. Pattern recognition is done by adding a layer of for instance supervised ML method. On the contrary, CNNs are mainly supervised with back-propagation for training and feed-forward for classification and formulation of similar belief states. For the DBNs as stated before, is greedy layer-wise unsupervised training with an extension of higher layers so as to perform the classification tasks.

It is valuable to mention that the DeSTIN is a type of *pyramid-like* approach like Laplacian or Gaussian pyramids [Adelson et al., 1984; Bister et al., 1990; Burt et al., 1981] with significant differences. Interested reader is referred to [Karnowski, 2012] for more detailed discussions around the differences.

5.5 Sequential DBNs

Conditional random fields (CRFs) are suitable for sequential labeling [Lafferty et al., 2001], they are used to represent the conditional dependencies as discussed in chapter 2. However, they have to be carefully designed by the engineer regarding the task at hand, which is time consuming and difficult task to do. It is also shown that longer-range dependencies in tasks such as speech is quite useful [Hermansky and Sharma, 1999]. In [Gunawardana et al., 2005; Sung and Jurafsky, 2009; Yu et al., 2010b; Yu and Deng, 2010] we can find several methods and approaches to add hidden layers to CRFs to extract the features which are not explicit in data. DBNs are the static case of these methods [Andrew and Bilmes, 2012].

In this section we introduce the *sequential RBM* (*SRBM*), see Figure 5.4. As one may expect it is similar to the ordinary RBM with a great difference that it spreads over time. Assume the visible input matrix along time $\boldsymbol{V} \in \mathbb{R}^{n_v \times T}$ with T time slice dependency, and the hidden layer matrix as $\boldsymbol{H} \in \mathbb{R}^{n_h \times T}$. All visible layer variables are independent given the hidden layer, and all rows of the hidden layer are independent given the visible layer [Andrew and Bilmes, 2012].

As usual a *sequential DBN* (*SDBN*) is formed by stacking multiple block of SRBN on top of each other, as you can see in Figure 5.4, however there is another possibility to think of SDBN as a serial concatenation of a set of DBNs along time instances, time is an additional dimension to the structure.

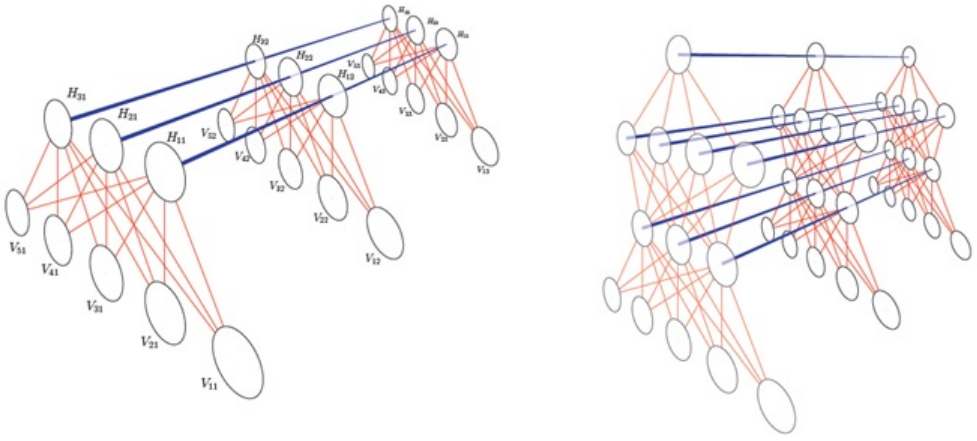


FIGURE 5.4: Sequential RBM (left) and Sequential DBN (right). Both consist of $T = 3$ time frames, and have $n_1 = 5$ input units and $n_2 = 3$ hidden units per frame in the first layer. The SDBN has $n_3 = 4$ hidden units per frame in the second layer, plus a single multinomial output per frame. The red edges correspond to the weights of the matrix $\boldsymbol{W}_0$, while the blue edges have weights given by $\boldsymbol{t}$. Edges across layers between adjacent time frames corresponding to $\boldsymbol{W}_\delta$ (e.g., from $\boldsymbol{V}_{11}$ to $\boldsymbol{H}_{12}$) are omitted from the figures for clarity [Andrew and Bilmes, 2012].

Starting with the input at layer 1 to the output layer indexed as L. For the hidden layers in between $l = 2, , L-1$, the hidden layer matrix is a binary matrix $\boldsymbol{H}^l \in \{\pm 1\}^{n_l \times T}$ where n_l is the number of units in each layer and T is the length along time dimension, with weight matrices $\boldsymbol{W}_\delta^l$ and transition parameters $\boldsymbol{t}^l \in \mathbb{R}^{n_l}$ along the time dimension (interaction between adjacent frames within each row of $\boldsymbol{H}^l$, note that there is no interaction between the visible layers along the time frames). δ is the interaction between different layers of multiple time frames, hidden-visible and hidden-hidden as you can see in 5.5 for the SRBM.

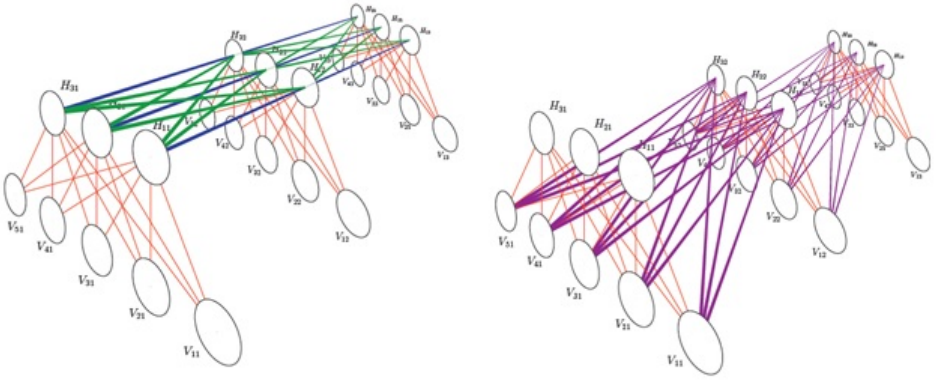


FIGURE 5.5: Interacted Sequential RBMs [Andrew, 2009].

For the training of an SDBN we have to first train each SRBM with CD algorithm, layer-by-layer. In [Andrew and Bilmes, 2012], a fine-tuning process has been done using the *stochastic gradient descent*, and could be computed via dynamic programming. The SDBN is implemented on the TIMIT phone recognition data set. With the experimental setup explained in [Andrew and Bilmes, 2012], and the best result obtained by a 150 units/frame, 8 layers, $\delta_{max} = 1$, where δ_{max} is the maximum number of links between a hidden and visible unit along different time frames. Although, this achievement outperforms many currently used methods, it is not as good as the performance of the architecture introduced in [Dahl et al., 2010] which is the mcRBM.

There are some similarities between these models and the TRBMs or conditional DBNs. TRBM architecture is more general than the SDBN, however the interacted SDBN covers broader concept of time dependencies, and they all differ from conditional DBNs, since CDBNs have links between visible units as well.

5.6 Recurrent Neural Networks

The *recurrent neural networks* (*RNNs*) are good at modeling dynamic characteristics of data, they are able to memorize and remember the context information due to their

recurrent schematic. There are many variants of recurrent neural networks such as *Elman networks* [Elman, 1990], *Jordan networks* [Jordan, 1986], *time delay neural networks* [Lang et al., 1990], and *echo state networks* [Jaeger, 2001]. Due to their difficulty involved in training this kind of network, they had become obsolete, however recent advances in *Hessian-free* optimization have succeeded in resurrecting their their deserving place among other neural network architectures [Sutskever et al., 2011].

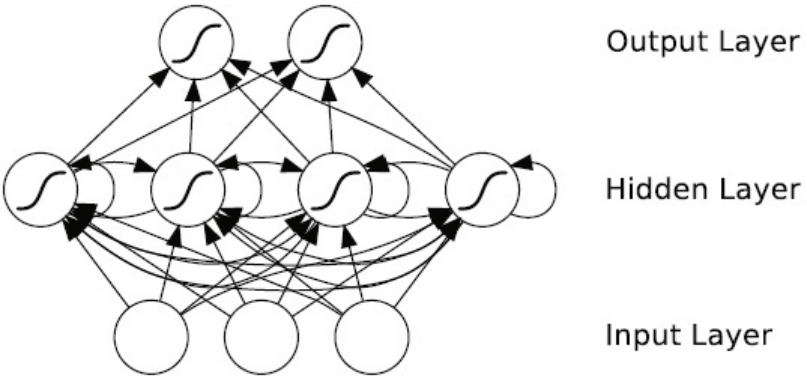


FIGURE 5.6: A standard recurrent neural network [Graves, 2012].

There is a useful way to visualize the RNNs architecture, called *unfolding*. As illustrated in Figure 5.7, there is no cycle in the unfolded graph. This new schematic (unfolded graph) make the visualization of more complex architectures, in terms of update dependencies, easier. It is sometimes of great interest to incorporate information of both past and future instances along a sequence. For instance, for the classification task of a written letter, it is informative if we know the letters after it as well as the ones before. This new information are included in a network called *bidirectional recurrent neural networks* (*BRNNs*), see Figure 5.8. The BRNNs outperform many *unidirectional* RNN architectures, you can find some of them experiments in [Schuster, 1999; Fukada et al., 1999; Chen and Chaudhari, 2004; Baldi et al., 2001]. For more detailed discussion about bidirectional RNNs you can refer to [Graves, 2012; Schuster, 1999; Schuster and Paliwal, 1997; Baldi et al., 1999].

The RNN architecture introduced in [Vinyals et al., 2012], follows the formulation in [Sutskever et al., 2011]:

$$\boldsymbol{h}_t = tanh(\boldsymbol{W}_{hx}\boldsymbol{x}_t + \boldsymbol{W}_{hh}\boldsymbol{h}_{t-1})\boldsymbol{o}_t = softmax(\boldsymbol{W}_{oh}\boldsymbol{h}_t) \tag{5.1}$$

where the bias terms are omitted for simplicity, $\boldsymbol{h}_t$ represents the hidden state of the network at time t, and $\boldsymbol{W}_{hx}$, $\boldsymbol{W}_{hh}$ and $\boldsymbol{W}_{oh}$ are parameters to be learned. Note that, due to the recursion over time on $\boldsymbol{h}_t$, the RNNs can be seen as a very deep network with T layers, where T is the number of time steps. We define the initial seed of the network

$\boldsymbol{h}_0$ to be another parameter of our model, and we optimize the cross-entropy between the predicted phone posterior output $\boldsymbol{o}_t$, and the true target, similar to how DNNs and MLPs are trained.

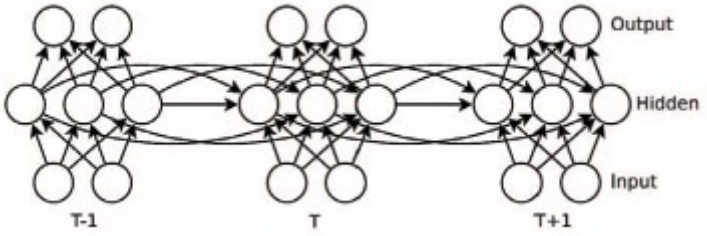


FIGURE 5.7: The unfolded architecture of a recurrent neural network. An RNN is a very deep feed-forward neural network whose weights are shared across time. The nonlinear activation function used by the hidden units is the source of the RNN's rich dynamics. Note that the same weights are reused at every time step. Bias weights are omitted for clarity [Vinyals et al., 2012].

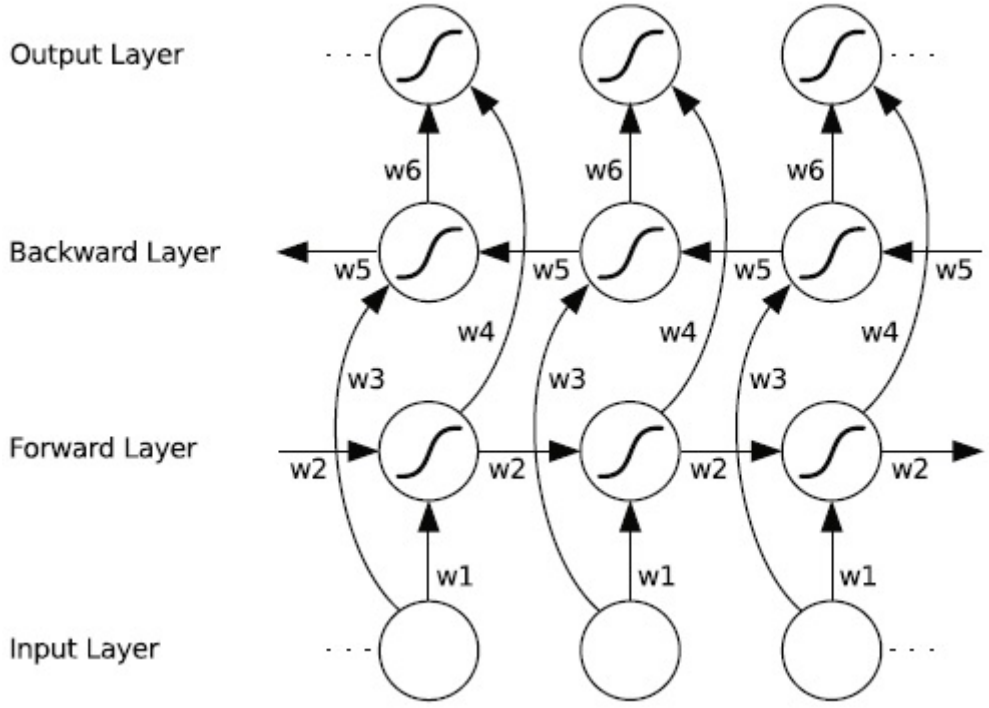


FIGURE 5.8: The unfolded architecture of a bidirectional recurrent neural network. Six distinct sets of weights are reused at every timestep, corresponding to the input-hidden, hidden-hidden and hidden-output connections of the two hidden layers. Note that no information flows between the forward and backward hidden layers; this ensures that the unfolded graph is acyclic [Graves, 2012].

5.7 Deep Long-Short Term Memory Recurrent Neural Networks

The recurrent neural networks are a natural choice for representing sequential and contextual information through the patterns. However, for the ordinary RNNs the range of contextual information is quite limited regarding to a phenomenon called *vanishing gradient problem* [Graves, 2012; Hochreiter et al., 2001; Bengio et al., 1994]. As illustrated in Figure 5.9 error gradients vanish quickly in an exponential fashion with respect to the size of the time lag between important events. Several attempts were made during 1990s, including non-gradient based training algorithms [Wierstra et al., 2005; Schmidhuber et al., 2007] such as *simulated annealing* and *discrete error propagation* [Bengio et al., 1994], *explicitly introduced time delays* [Lang et al., 1990; Lin et al., 1996; Plate] or *time constants* [Mozer, 1991] and *hierarchical sequence compression* [Schmidhuber, 1992], to find a solution for the problem. Another important method, which is the subject of this section, is the *long short-term memory* (*LSTM*) architecture [Hochreiter and Schmidhuber, 1997].

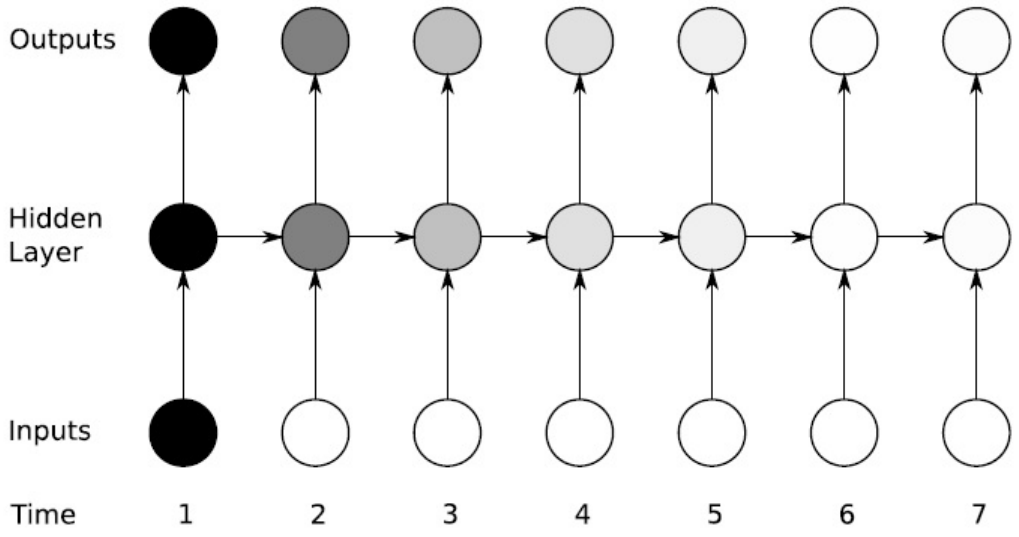


FIGURE 5.9: Vanishing Gradient Problem for RNNs. The shading of the nodes in the unfolded network indicates their sensitivity to the inputs at time one (the darker the shade, the greater the sensitivity). The sensitivity decays over time as new inputs overwrite the activations of the hidden layer, and the network *forgets* the first inputs [Graves, 2012].

The LSTM architecture comprises a set of recurrently linked blocks, one may think of a block as a small subnet which is similar to a differentiable memory chip in a digital computer. In the architecture used in [Graves, 2012], each block consists of one or more self-connected *memory cells* three multiplicative units-the *input*, *output* and *forget gates*-that provide continuous analogues of *write*, *read* and *reset* operations for the memory

cells, see Figure 5.10. As it is depicted in Figure 5.11, replacing the standard hidden units of an RNN by LSTM blocks, an LSTM network is achieved.

Note that in the original LSTM block, there was no forget gate [Gers et al., 2000]. It was considered, along with the respective weights [Gers et al., 2003], later to produce the *extended LSTM* [Gers, 2001]. These gates together with the peephole weights provide the capability to *smartly* learn, what information to store, how long to keep it and when to read it out. One of the most important advantages of LSTM networks to many other models such as Markov models, is their ability to have variable dependencies. For instance in a second-order Markov chain the current state is conditioned only on two previous steps, however in an LSTM network it is possible to have different dependencies in different cases.

These multiplicative gates allow the LSTM blocks to memorize information for a long time and access them whenever it is needed. This helps to mitigate the vanishing gradient problem. As an example to make it clear, assume that the input gate is closed (i.e. has an activation near 0), therefore this block does not accept any new input so the current input can be available in the block and through the network much later by opening the output gate [Graves, 2012].

The architectures using LSTM blocks, are implemented in many different tasks which require long range dependencies, such as learning context free language [Gers and Schmidhuber, 2001], recalling high precision real numbers over extended noisy sequences [Hochreiter and Schmidhuber, 1997] and various tasks requiring precise timing and counting [Gers et al., 2003], protein secondary structure prediction [Hochreiter et al., 2007], music generation [Eck and Schmidhuber], reinforcement learning [Bakker, 2001], speech recognition [Graves and Schmidhuber, 2004; Graves and Fernández, 2006] and handwriting recognition [Liwicki et al., 2007; Graves and Liwicki, 2007].

An approximate error gradient which is obtained by a *real time recurrent learning* [Robinson and Fallside, 1987] incorporated with a *back-propagation through time* (*BPTT*) [Williams and Zipser, 1995] is employed as the original learning algorithm for LSTM networks [Hochreiter and Schmidhuber, 1997]. There are two possibilities to implement the BPTT, one the truncated version which is an approximation and the other is the exact gradient (untruncated) BPTT [Graves and Schmidhuber, 2004]. In the truncated version, BPTT is terminated after one time step, with the truncating gradient it is possible to the algorithm totally online (the weights update can be done in every time step). However the untruncated one is easier to debug [Graves, 2012].

It is easy to develop new variants of these memory blocks, due to their simple inherent architecture which are composed of multiplicative and summation units. One can find

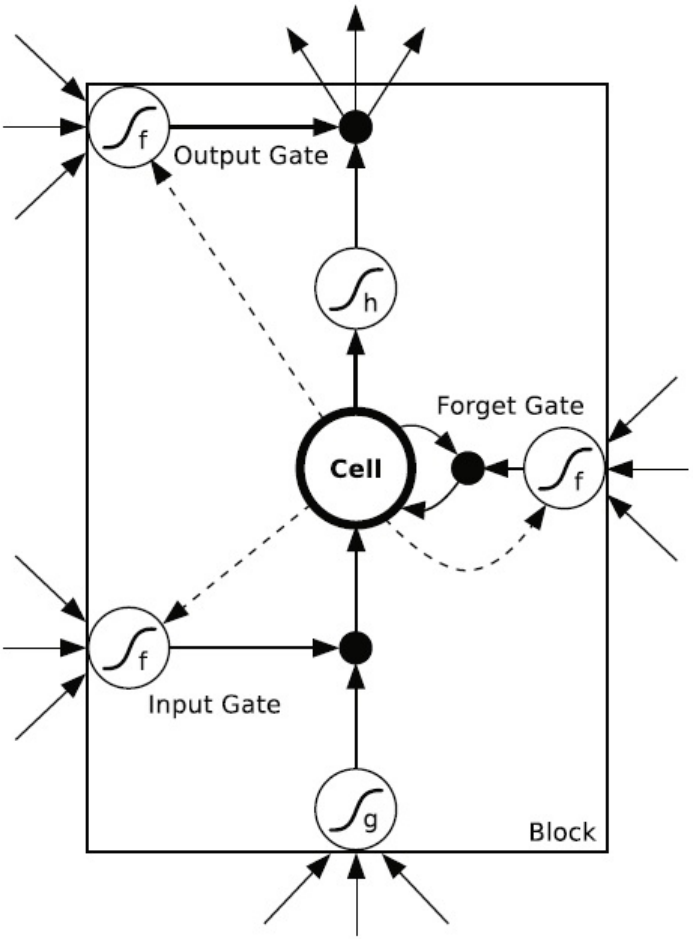


FIGURE 5.10: An LSTM memory block with one cell. The three gates are nonlinear summation units that collect activations from inside and outside the block, and control the activation of the cell via multiplications (small black circles). The input and output gates multiply the input and output of the cell while the forget gate multiplies the cell's previous state. No activation function is applied within the cell. The gate activation function f is usually the logistic sigmoid, so that the gate activations are between 0 (gate closed) and 1 (gate open). The cell input and output activation functions (g and h) are usually *tanh* or logistic sigmoid, though in some cases h is the identity function. The weighted *peephole* connections from the cell to the gates are shown with dashed lines. All other connections within the block are unweighted (or equivalently, have a fixed weight of 1.0). The only outputs from the block to the rest of the network emanate from the output gate multiplication [Graves, 2012].

some of these variants in [Bayer and Wierstra, 2009] for the tasks such as learning the context-free and context-sensitive languages.

Using LSTM as the network architecture in a *bidirectional* recurrent neural network yields bidirectional LSTM [Graves and Schmidhuber, 2004; Graves and Schmidhuber; Chen and Chaudhari, 2005; Thireou and Reczko, 2007]. Bidirectional LSTM provides access to long range context in both input directions. For more detailed discussion about the bidirectional concept and its relevant subjects I recommend [Graves, 2012] to the readers.

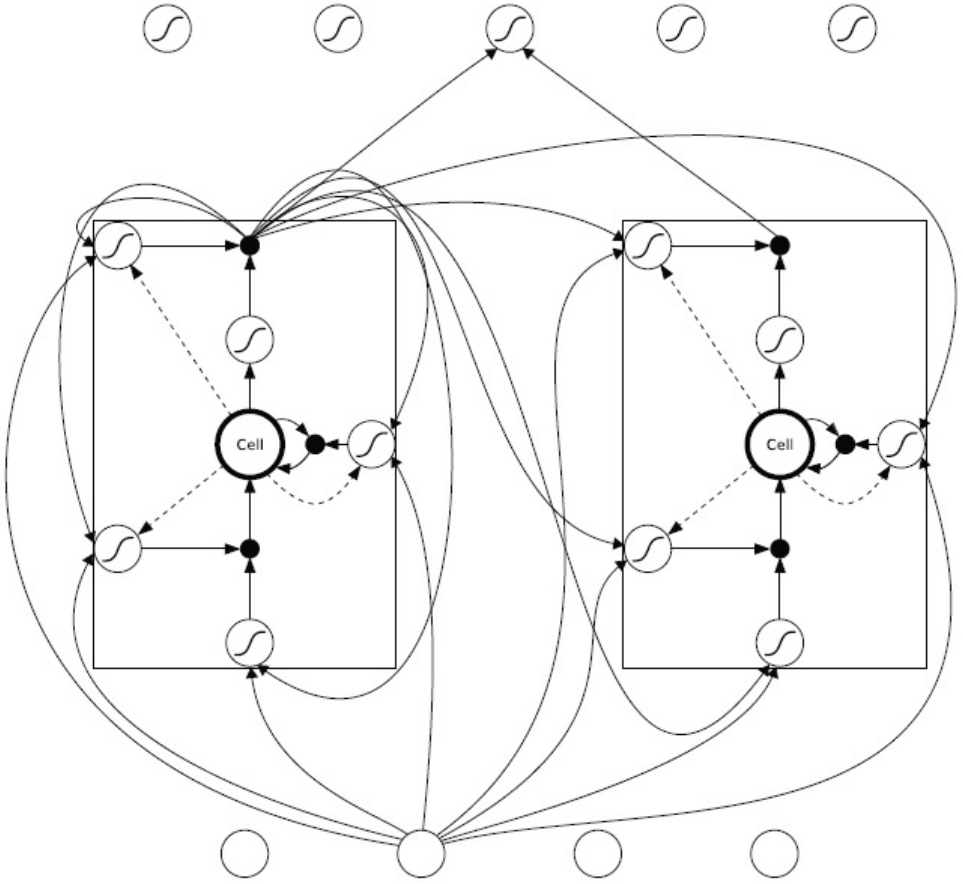


FIGURE 5.11: An LSTM network. The network consists of four input units, a hidden layer of two single-cell LSTM memory blocks and five output units. Not all connections are shown. Note that each block has four inputs but only one output [Graves, 2012].

5.8 HMM-LSTM

The context of hybrid architecture is usually predicated on the use of MLPs incorporated with HMMs. However, there is a considerable interest in using the HMMs in combination with RNNs [Robinson, 1994; Neto et al., 1995; Kershaw et al., 1995; Senior and Robinson, 1996]. Their advantages over MLPs remained inconclusive in early work [Robinson et al., 1993].

In [Graves, 2012], you can find a comparison between the performance of standard HMMs with and without context dependent phoneme models, and the hybrid architectures using BLSTM, LSTM, and BRNNs on the TIMIT speech corpus, in order to asses the potential of LSTM and BLSTM for hybrid HMM-ANN systems. Both context-dependent and context-independent are outperformed by HMM-BLSTM hybrid model. The best result was achieved with the HMM-BLSTM hybrid using a weighted error signal. This is what we would expect, since the effect of error weighting is to make

all phonemes equally significant, as they are to the phoneme error rate. Note that the hybrid systems had considerably fewer free parameters than the context-dependent HMM. This is a consequence of the high number of states required for HMMs to model contextual dependencies [Graves, 2012].

5.9 Hierarchical Temporal Memory

Hierarchical Temporal Memory (*HTM*) is an architecture recently proposed based on concepts described in [Hawkins and Blakeslee, 2004] about neocortex [Arel et al., 2010]. There are similarities between this new model and neural networks, however they are not quite the same. HTM is more relative to deep architectures [Maltoni, 2011]. HTM comprises of a tree-shaped network of nodes as you can see in Figure 5.12, however these nodes are not the units in a neural network, rather they are elements which discover the causes of their inputs, passes beliefs up and the predictions down the network. Each node stores common sequences, forms stable beliefs (invariant representation) at top by changing sensory data, and forms changing the sensory predictions using the stable beliefs at top [Numenta, 2011].

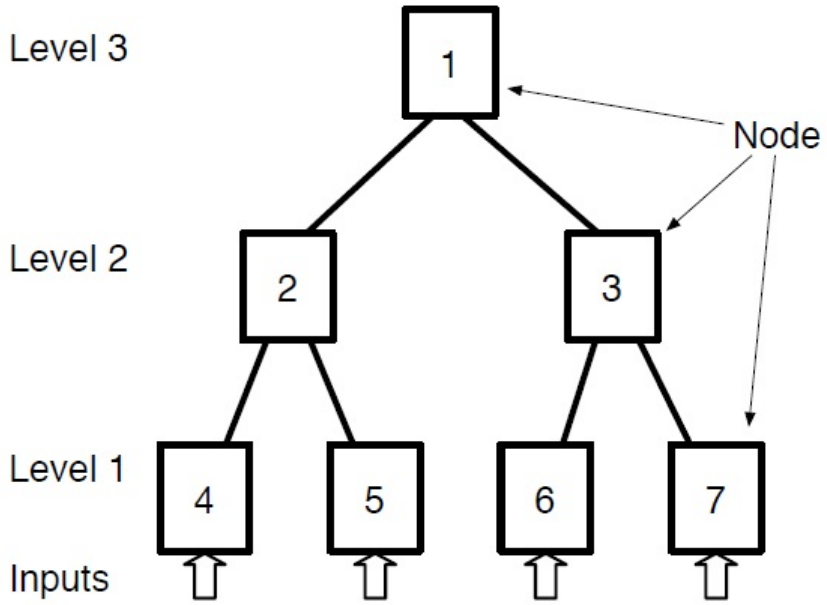


FIGURE 5.12: A simple HTM network that has 7 nodes arranged in a 3 level hierarchy. The levels are numbered from level one to level three, with level one at the bottom of the hierarchy and level three at the top. Inputs are fed to the nodes at the bottom level of the hierarchy [George, 2008].

An HTM network comprises of regions (levels) in a hierarchical order. This hierarchical structure results in decrease of training time, memory usage, and also introducing a from

of generalization. The nodes in each level or region are the basic component and memory module, containing algorithms which are similar in nature [George, 2008]. Ascending to the upper levels, the number of nodes in each region often decreases. Higher levels of the hierarchy are able to reuse the patterns learned in the lower levels. They use the same philosophy of human brain, for instance in the vision process, at lowest level the brain stores information about simple structures of the visual field such as edges and corners, then in the upper levels these information is used to produce mid-level representations such as curves or textures, these are more complex structures. It is possible to think of an arc as an edge of an ear, or the top of a steering wheel. If we continue this combining of the simple component to more structured ones, we may conclude to high-level objects, such as heads, houses, or cars. It is important to learn a new component in the higher level we don't need to learn everything from the beginning, and also we can do the generalization (prediction) to other unexperienced situations, see Figure 5.13.

In the learning phase, in the very first level, nodes extract the most common patterns (such as edges in the example described before) and assign indices to them. Then the temporal dependencies between one input sequence to another, are modeled by means of probabilistic interpretations and clustered. After this level it continues to the upper levels to construct more complex representations from the simple components. In the top-down pass, the characterizations of the higher levels are fed back to the lower ones. After the training, the recognition is done via a Bayesian belief propagation algorithm [Pearl, 1988], this identifies the most probable pattern as the output of the network which is called belief [George, 2008]. There are other variants of the HTM in different literature such as [Miller and Lommel, 2006; Behnke, 2003].

In 2005 Numenta, Inc. was formed in California, as they stated their goals themselves, to *develop biologically-inspired machine intelligence technology for both commercial and scientific use.* For more information such as white papers, technical issues, developments, and new release of their software you can visit [Inc.].

5.10 Conclusions

The concept of deep learning and its preliminaries with the most prominent architectures have been presented so far. The differences have been shown and the comparisons noted. However it is not possible to compare all the methods, since they have been implemented in different tasks and on different data sets. Sometimes a method is the successor of another architecture in order to circumvent some of the shortcomings of its predecessor(s), however it is not always met, which makes the comparisons even more unattainable.

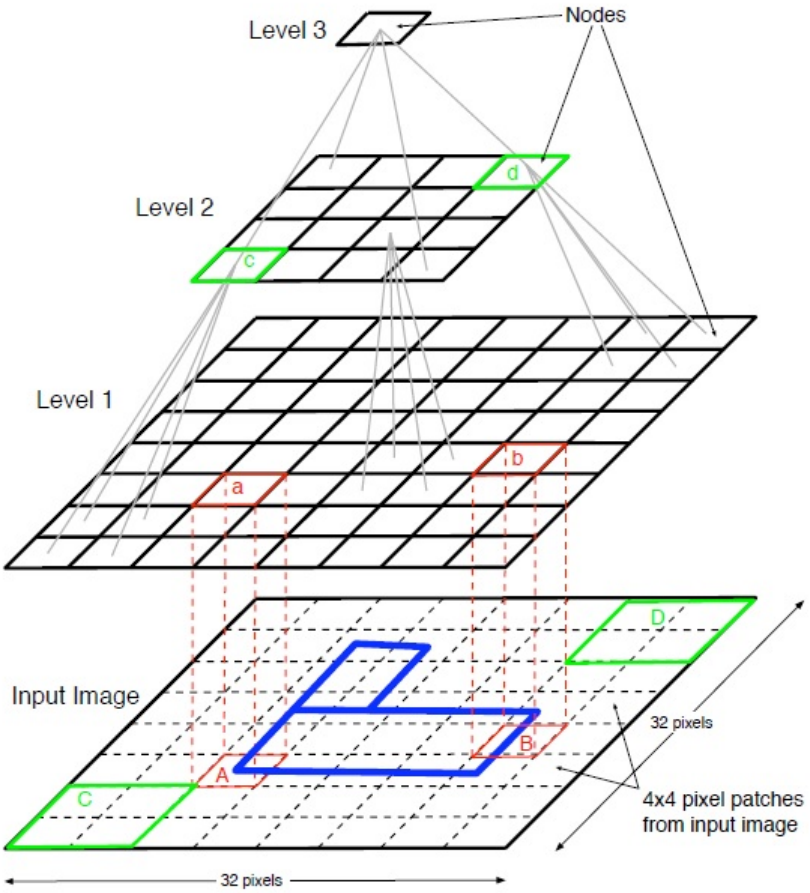


FIGURE 5.13: Structure of the HTM network for learning invariant representations for the binary images world. Level one of the network has 64 nodes arranged in an 8×8 grid. Each node is shown as a square. The input to a level-2 node comes from the outputs of 4 level-1 nodes as marked. The outputs from all the level-2 nodes go to a single level-3 node. Input is fed to the nodes at level-1. The input image, a frame from the training videos, is of size 32 pixels by 32 pixels. This image is divided into adjoining patches of 4 pixels by 4 pixels as shown. Each level-1 node's input corresponds to one such 4×4 patch. Two nodes at level-1 are marked (a) and (b). Squares marked (A) and (B) in the input image represent the respective receptive fields of these nodes. Each level-1 node *sees* only a small portion of the input image. The squares marked by (C) and (D) correspond to the receptive fields of nodes at level-2 marked (c) and (d) [George, 2008].

Although, there are several other methods introduced, the DBN is still the promising and the mainstream for the case of static patterns. For the case of sequential patterns, the situation is slightly different, sequential DBNs play an important role, however the recurrent networks are still of great interests. For instance in phoneme recognition on TIMIT data set, to the best of my knowledge the LSTM network is the state-of-the-art in this field, see Table 5.1.

As it is shown in the table the LSTM networks with transducer or with connectionist temporal classification outperforms the previous architectures [Graves et al., 2013].

Method	PER
Sequential DBN	24.2%
DBN-HMM (Hybrid Model)	23.0%
DBN with mcRBM feature extraction (Hybrid Model)	20.5%
Deep Long-Short Term Memory RNNs with CTC	18.4%
Deep Long-Short Term Memory RNNs with transducer	**17.7%**

TABLE 5.1: A comparison of different architectures performed on TIMIT data set for phoneme recognition task in terms of phoneme error rate (PER) on the core test set. The deep LSTM RNN with transducer has the best performance [Graves et al., 2013].

There are a huge amount of different methods for exploiting deep networks in different kinds of tasks related to pattern recognition, even for the phoneme recognition one may find dozens of methods, which needs plenty of pages just naming them while the performances are in the same range or even worse, however it has been tried to show and mention the mainstream in this area.

This LSTM network, which is pioneered by Hinton's group as many other architectures, proposed also working on Large Vocabulary Speech Recognition or combining frequency domain convolutional neural networks with deep LSTM as another interesting research direction. However, that would be an interesting idea to implement this method on other areas of audio processing such as music genre recognition, gender recognition, music labeling. Also it is possible to try different algorithm for decoding in combination with variety of loss functions and optimization methods. Notice that this LSTM networks are mostly used in supervised labeling and less attempts have been made on the tasks involving unsupervised paradigm of learning, for instance as the building blocks of an auto-encoder architecture.

Bibliography

RO Duda, PE Hart, and DG Stork. *Pattern classification*. 2012.

Daphne Koller and Nir Friedman. *Probabilistic graphical models*. 2009. ISBN 9780262013192.

Michael Alder. *An Introduction to Pattern Recognition*. 1997.

Paul Viola and MJ Jones. Robust real-time face detection. *International journal of computer vision*, 57(2):137–154, 2004.

CM Bishop. *Pattern recognition and machine learning*. 2006. ISBN 9780387310732.

L Wassermann. *All of statistics*. Springer, 2003. ISBN 0387402721.

B Everitt and A Skrondal. *The Cambridge dictionary of statistics*. 4th editio edition, 2010. ISBN 9780521766999.

Stellan Ohlsson. *Deep learning: How the mind overrides experience*. 2011. ISBN 9780521835688.

Randall C O'Reilly and Yuko Munakata. *Computational Explorations in Cognitive Neuroscience*. The MIT Press, 2000. ISBN 0262650541. URL http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20&path=ASIN/0262650541.

L Tarassenko, P Hayton, N Cerneaz, and M Brady. Novelty detection for the identification of masses in mammograms. *... Neural Networks, 1995 ...*, (x), 1995. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=497860.

Michael I. Jordan and Christopher M. Bishop. Neural Networks. (1562), 1996.

JF Roddick and M Spiliopoulou. A survey of temporal knowledge discovery paradigms and methods. *Knowledge and Data Engineering ...*, 14(4):750–767, 2002. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1019212.

J. Whittaker. *Graphical Models in Applied Multivariate Statistics*. Wiley, 1990. ISBN 978-0-471-91750-2.

Michael I. Jordan. *Learning in Graphical Models*. MIT Press, 1999. ISBN 9780262600323.

S. L. Lauritzen. *Graphical Models*. Oxford University Press, 1996. ISBN 978-0198522195.

R. G. Cowell, P. Dawid, S. L. Lauritzen, and D. J. Spiegelhalter. *Probabilistic Networks and Expert Systems*. Springer, 1999. ISBN 978-0-387-22630-9.

Hervé a. Bourlard and Nelson Morgan. *Connectionist Speech Recognition*. Springer US, Boston, MA, 1994. ISBN 978-1-4613-6409-2. doi: 10.1007/978-1-4615-3210-1.

E De Bono. *The mechanism of mind*. 1969. URL http://onlinelibrary.wiley.com/doi/10.1111/j.1469-8749.1970.tb01928.x/abstract.

J. W. Kalat. *Biological psychology*. Wadsworth, Cengage Learning, 10 edition, 2009. ISBN 978-0-495-60300-9. URL http://psycnet.apa.org/psycinfo/1996-98040-000.

D. E. Rumelhart and J. L. McClelland. Parallel distributed processing. *Psychological and biological models*, 1, 1987. URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.164.3258&rep=rep1&type=pdf.

JL McClelland and DE Rumelhart. Parallel distributed processing. *Psychological and biological models*, 2, 1987. URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.164.3258&rep=rep1&type=pdf.

JA Anderson. *An introduction to neural networks*. 1995. URL http://www.mitpressjournals.org/doi/pdf/10.1162/jocn.1996.8.4.383b.

H R Wilson and J D Cowan. Excitatory and inhibitory interactions in localized populations of model neurons. *Biophysical journal*, 12(1): 1–24, January 1972. ISSN 0006-3495. doi: 10.1016/S0006-3495(72)86068-5. URL http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=1484078&tool=pmcentrez&rendertype=abstract.

S Grossberg. Adaptive pattern classification and universal recoding: I. Parallel development and coding of neural feature detectors., July 1976. ISSN 0340-1200. URL http://www.ncbi.nlm.nih.gov/pubmed/974165.

RA Chapman. The repetitive responses of isolated axons from the crab, Carcinus maenas. *Journal of Experimental Biology*, 45:475–488, 1966. URL http://jeb.biologists.org/content/45/3/475.short.

HJ Kappen. An introduction to stochastic neural networks. *Handbook of Biological Physics*, 2001. URL http://www.sciencedirect.com/science/article/pii/S1383812101800166.

S Samarasinghe. *Neural networks for applied sciences and engineering: from fundamentals to complex pattern recognition*. Auerbach Publications, Taylor & Francis Group, Boca Raton, Florida, 2007. ISBN 0-8493-3375-X. URL http://books.google.com/books?hl=en&lr=&id=EyFeUiJibooC&oi=fnd&pg=PR17&dq=Neural+Networks+for+Applied+Sciences+and+Engineering:+From+Fundamentals+to+Complex+Pattern+Recognition&ots=59O0f_P2Su&sig=1wovCsJb6OaOHVgZOqfZKvfoiYM.

Anthony Zaknich. *Neural Networks for Intelligent Signal Processing*. World Scientific Publishing Co. Pte. Ltd., 2003. ISBN 981-238-305-0.

Ben Kröse and Patrick Van der Smaft. *An introduction to neural networks*. Number November. 8 edition, 1996. URL http://www.mitpressjournals.org/doi/pdf/10.1162/jocn.1996.8.4.383b.

Nikhil Garg and James Henderson. Temporal Restricted Boltzmann Machines for Dependency Parsing. *ACL (Short Papers)*, 2011. URL http://cui.unige.ch/~garg/publications/garg-acl2011-slides.pdf.

GE Hinton and TJ Sejnowski. Optimal perceptual inference. *Proceedings of the IEEE conference on ...*, pages 0–5, 1983. URL http://www.csri.utoronto.ca/~hinton/absps/optimal.pdf.

Samy Bengio, Li Deng, Hugo Larochelle, Honglak Lee, and Ruslan Salakhutdinov. Guest Editors' Introduction: Special Section on Learning Deep Architectures. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1795–1797, August 2013. ISSN 0162-8828. doi: 10.1109/TPAMI.2013.118.

Jonathan Laserson. From Neural Networks to Deep Learning. *XRDS: Crossroads, The ACM Magazine for Students*, 18(1):29, September 2011. ISSN 15284972. doi: 10.1145/2000775.2000787. URL http://dl.acm.org/citation.cfm?doid=2000775.2000787.

David Marr. Vision: A computational investigation into the human representation and processing of visual information, Henry Holt and Co. *Inc., New York, NY*, 1982.

Li Deng. Three Classes of Deep Learning Architectures and Their Applications: A Tutorial Survey. *research.microsoft.com*, 2013.

Yann Lecun and Marc'Aurelio Ranzato. Deep Learning Tutorial, ICML, Atlanta, 2013.

P Koehn. *Statistical machine translation*. Cambridge University Press, 2009. ISBN 9780521874151. URL http://www.google.com/patents?hl=en&lr=

&vid=USPAT7624005&id=GpzKAAAAEBAJ&oi=fnd&dq=Statistical+Machine+Translation&printsec=abstract.

FQ Lauzon. An introduction to deep learning. *Information Science, Signal Processing and their ...*, 2012. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6310529.

Honglak Lee, Peter Pham, and Andrew Y Ng. Unsupervised feature learning for audio classification using convolutional deep belief networks. pages 1–9, 2009a.

Philippe Hamel and Douglas Eck. Learning Features from Music Audio with Deep Belief Networks. *ISMIR*, (Ismir):339–344, 2010. URL http://musicweb.ucsd.edu/~sdubnov/Mu270d/DeepLearning/FeaturesAudioEck.pdf.

Christian Plahl and TN Sainath. Improved pre-training of deep belief networks using sparse encoding symmetric machines. *..., Speech and Signal ...*, (2):4165–4168, 2012. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6288836.

GE Hinton, S Osindero, and YW Teh. A fast learning algorithm for deep belief nets. *Neural computation*, 1554:1527–1554, 2006. URL http://www.mitpressjournals.org/doi/abs/10.1162/neco.2006.18.7.1527.

Z You, X Wang, and B Xu. INVESTIGATION OF DEEP BOLTZMANN MACHINES FOR PHONE RECOGNITION. *danielpovey.com*, pages 7600–7603, 2013. URL http://www.danielpovey.com/files/ICASSP13/pdfs/0007600.pdf.

Ruslan Salakhutdinov and Geoffrey Hinton. Deep boltzmann machines. *International ...*, (3):448–455, 2009. URL http://machinelearning.wustl.edu/mlpapers/paper_files/AISTATS09_SalakhutdinovH.pdf.

Ruslan Salakhutdinov and Hugo Larochelle. Efficient learning of deep boltzmann machines. *International ...*, 9, 2010. URL http://machinelearning.wustl.edu/mlpapers/paper_files/AISTATS2010_SalakhutdinovL10.pdf.

Geoffrey Hinton and Ruslan Salakhutdinov. A better way to pretrain deep Boltzmann machines. *Advances in Neural ...*, (3):1–9, 2012. URL http://machinelearning.wustl.edu/mlpapers/paper_files/NIPS2012_1178.pdf.

Yoshua Bengio and Y LeCun. Scaling learning algorithms towards AI. *Large-Scale Kernel Machines*, (1):1–41, 2007. URL http://www.iro.umontreal.ca/~lisa/bib/pub_subject/language/pointeurs/bengio+lecun-chapter2007.pdf.

Itamar Arel, Derek Rose, and Robert Coop. Destin: A scalable deep learning architecture with application to high-dimensional robust pattern recognition. *Proc. AAAI*

Workshop on Biologically Inspired ..., 2009a. URL http://www.aaai.org/ocs/index.php/FSS/FSS09/paper/download/951/1268DeSTIN.

Y LeCun and L Bottou. Gradient-based learning applied to document recognition. *Proceedings of the ...*, 1998. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=726791.

Y LeCun and Y Bengio. Convolutional networks for images, speech, and time series. *...handbook of brain theory and neural networks*, 1995. URL http://www.iro.umontreal.ca/labs/neuro/pointeurs/handbook-convo.pdf.

DH Hubel and TN Wiesel. Receptive fields and functional architecture of monkey striate cortex. *The Journal of physiology*, pages 215–243, 1968. URL http://jp.physoc.org/content/195/1/215.short.

JL McClelland, J Feldman, and B Adelson. Connectionist models and cognitive science: Goals, directions and implications. *...to the National Science ...*, 1986. URL http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:Connectionist+Models+and+Cognitive+Sceience:+Goals,+Directions+and+Implications#1.

GE Hinton. Connectionist learning procedures. *Artificial intelligence*, 40(1-3): 185–234, September 1989. ISSN 00043702. doi: 10.1016/0004-3702(89)90049-0. URL http://linkinghub.elsevier.com/retrieve/pii/0004370289900490http://www.sciencedirect.com/science/article/pii/0004370289900490.

Paul E Utgoff and David J Stracuzzi. Many-layered learning. *Neural computation*, 14(10):2497–529, October 2002. ISSN 0899-7667. doi: 10.1162/08997660260293319. URL http://www.ncbi.nlm.nih.gov/pubmed/12396572.

Pascal Vincent, H Larochelle, and I Lajoie. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *...of Machine Learning ...*, 11:3371–3408, 2010. URL http://dl.acm.org/citation.cfm?id=1953039.

Johan Hastad. Almost optimal lower bounds for small depth circuits. *Proceedings of the eighteenth annual ACM symposium ...*, 1986. URL http://dl.acm.org/citation.cfm?id=12132.

Johan Hastad and Mikael Goldmann. On the power of small-depth threshold circuits. *Computational Complexity*, 1(2):113–129, June 1991. ISSN 1016-3328. doi: 10.1007/BF01272517. URL http://link.springer.com/10.1007/BF01272517.

Yoshua Bengio. Learning deep architectures for AI. *Foundations and trends in Machine Learning*, 2(1):1–127, 2009. ISSN 1935-8237. doi: 10.1561/2200000006. URL http://www.nowpublishers.com/product.aspx?product=MAL&doi=2200000006http://dl.acm.org/citation.cfm?id=1658424.

Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle. Greedy layer-wise training of deep networks. *Advances in neural ...*, (1), 2007. URL http://books.google.com/books?hl=en&lr=&id=Tbn1l9P1220C&oi=fnd&pg=PA153&dq=Greedy+Layer-Wise+Training+of+Deep+Networks&ots=V2sbCgmmY0&sig=tbRubYP1V1pLDbUoMMToeUhXNJ0.

Marc'Aurelio Ranzato, Christopher Poultney, Sumit Chopra, and Yann LeCun. Efficient learning of sparse representations with an energy-based model. *Advances in neural ...*, 2006. URL http://machinelearning.wustl.edu/mlpapers/paper_files/NIPS2006_804.pdf.

Hugo Larochelle, Yoshua Bengio, J Louradour, and Pascal Lamblin. Exploring strategies for training deep neural networks. *The Journal of Machine ...*, 1:1–40, 2009. URL http://dl.acm.org/citation.cfm?id=1577070.

Li Deng and Dong Yu. Deep convex net: A scalable architecture for speech pattern classification. *Proceedings of the Interspeech*, (August): 2285–2288, 2011a. URL http://www.truebluenegotiations.com/files/deepconvexnetwork-interspeech2011-pub.pdf.

L Deng and D Yu. Deep convex networks for image and speech classification. *ICML Workshop on Learning Architectures*, 2011b. URL http://research.microsoft.com/pubs/171500/ICML2011-Deng-Yu.docx.

Li Deng, Dong Yu, and John Platt. Scalable stacking and learning for building deep architectures. *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 2133–2136, March 2012. doi: 10.1109/ICASSP.2012.6288333. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6288333.

DH Wolpert. Stacked generalization. *Neural networks*, 87545(505), 1992. URL http://www.sciencedirect.com/science/article/pii/S0893608005800231.

Brian Hutchinson, Li Deng, and Dong Yu. Tensor deep stacking networks. *IEEE transactions on pattern analysis and machine intelligence*, pages 1–15, December 2012a. ISSN 1939-3539. doi: ACC5D5FE-AF7C-46B6-8570-3CE544BC8027.

Dong Yu and Li Deng. Accelerated Parallelizable Neural Network Learning Algorithm for Speech Recognition. *INTERSPEECH*, (August):

2281–2284, 2011. URL http://research.microsoft.com/pubs/152134/singlelayer4asr-interspeech2011-pub.pdf.

G Dahl, Marc'Aurelio Ranzato, Abdel-rahman Mohamed, and Geoffrey Hinton. Phone recognition with the mean-covariance restricted Boltzmann machine. *Advances in neural ...*, pages 1–9, 2010. URL http://machinelearning.wustl.edu/mlpapers/paper_files/NIPS2010_0160.pdf.

Marc'Aurelio Ranzato, A Krizhevsky, and GE Hinton. Factored 3-way restricted boltzmann machines for modeling natural images. *International ...*, 9:621–628, 2010a. URL http://machinelearning.wustl.edu/mlpapers/paper_files/AISTATS2010_RanzatoKH10.pdf.

Marc'Aurelio Ranzato and Geoffrey E. Hinton. Modeling pixel means and covariances using factorized third-order boltzmann machines. *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 2551–2558, June 2010. doi: 10.1109/CVPR.2010.5539962. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5539962.

B Hutchinson, L Deng, and D Yu. A deep architecture with bilinear modeling of hidden representations: Applications to phonetic recognition. *Acoustics, Speech and Signal ...*, (3):3–6, 2012b. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6288994.

GE Hinton and RR Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(July):504–507, 2006. URL http://www.sciencemag.org/content/313/5786/504.short.

GE Dahl, Dong Yu, Li Deng, and Alex Acero. Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition. *Audio, Speech, and ...*, 20(1): 30–42, 2012. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5740583.

Abdel-rahman Mohamed, George E. Dahl, and Geoffrey Hinton. Acoustic Modeling Using Deep Belief Networks. *IEEE Transactions on Audio, Speech, and Language Processing*, 20(1):14–22, January 2012. ISSN 1558-7916. doi: 10.1109/TASL.2011.2109382. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5704567.

QV Le, MA Ranzato, R Monga, Matthieu Devin, Kai Chen, Greg S. Corrado, Jeff Dean, and Andrew Y. Ng. Building high-level features using large scale unsupervised learning. *arXiv preprint arXiv: ...*, 2011. URL http://arxiv.org/abs/1112.6209.

AC Courville, James Bergstra, and Yoshua Bengio. A spike and slab restricted Boltzmann machine. *International ...*, 15:233–241, 2011a. URL http://machinelearning.wustl.edu/mlpapers/paper_files/AISTATS2011_CourvilleBB11.pdf.

TJ Mitchell and JJ Beauchamp. Bayesian variable selection in linear regression. *Journal of the American Statistical ...*, 1988. URL http://www.tandfonline.com/doi/abs/10.1080/01621459.1988.10478694.

Aaron Courville, James Bergstra, and Yoshua Bengio. Unsupervised models of images by spike-and-slab RBMs. *Proceedings of the ...*, 10, 2011b. URL http://machinelearning.wustl.edu/mlpapers/paper_files/ICML2011Courville_591.pdf.

Marc'Aurelio Ranzato, V Mnih, and GE Hinton. Generating more realistic images using gated MRF's. *... in Neural Information Processing ...*, pages 1–9, 2010b. URL http://machinelearning.wustl.edu/mlpapers/paper_files/NIPS2010_1163.pdf.

Charles Kemp, Amy Perfors, and Joshua B Tenenbaum. Learning overhypotheses with hierarchical Bayesian models. *Developmental science*, 10(3):307–21, May 2007. ISSN 1363-755X. doi: 10.1111/j.1467-7687.2007.00585.x. URL http://www.ncbi.nlm.nih.gov/pubmed/17444972.

AF Perfors and JB Tenenbaum. Learning to learn categories. *Annual Conference*, (1955):136–141, 2009. URL http://ebooks.adelaide.edu.au/dspace/handle/2440/58419.

Linda B Smith, Susan S Jones, Barbara Landau, Lisa Gershkoff-Stowe, and Larissa Samuelson. Object name learning provides on-the-job training for attention., January 2002. ISSN 0956-7976. URL http://www.ncbi.nlm.nih.gov/pubmed/11892773.

Fei Xu and Joshua B Tenenbaum. Word learning as Bayesian inference. *Psychological review*, 114(2):245–72, April 2007. ISSN 0033-295X. doi: 10.1037/0033-295X.114.2.245. URL http://www.ncbi.nlm.nih.gov/pubmed/17500627.

Ruslan Salakhutdinov, Joshua B Tenenbaum, and Antonio Torralba. Learning with Hierarchical-Deep Models. *IEEE transactions on pattern analysis and machine intelligence*, pages 1–14, December 2012. ISSN 1939-3539. doi: 67A1A5D7-FC14-405B-9802-CB5064DF6064. URL http://www.ncbi.nlm.nih.gov/pubmed/23267196.

Adam Coates, Blake Carpenter, Carl Case, Sanjeev Satheesh, Bipin Suresh, Tao Wang, David J. Wu, and Andrew Y. Ng. Text Detection and Character Recognition in Scene

Images with Unsupervised Feature Learning. pages 440–445, September 2011. doi: 10.1109/ICDAR.2011.95.

Honglak Lee, Roger Grosse, Rajesh Ranganath, and Andrew Y. Ng. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. *Proceedings of the 26th Annual International Conference on Machine Learning - ICML '09*, pages 1–8, 2009b. doi: 10.1145/1553374.1553453. URL http://portal.acm.org/citation.cfm?doid=1553374.1553453.

Yuanqing Lin, Z Tong, Shenghuo Zhu, and Kai Yu. Deep coding network. *Advances in Neural ...*, pages 1–9, 2010. URL http://machinelearning.wustl.edu/mlpapers/paper_files/NIPS2010_1077.pdf.

Marc Aurelio Ranzato, Y Boureau, and YL Cun. Sparse feature learning for deep belief networks. *Advances in neural information ...*, (Mcmc):1–8, 2007. URL http://machinelearning.wustl.edu/mlpapers/paper_files/NIPS2007_1118.pdf.

R Socher, CC Lin, Andrew Y. Ng, and Christopher D. Manning. Parsing natural scenes and natural language with recursive neural networks. *Proceedings of the ...*, 2011. URL http://machinelearning.wustl.edu/mlpapers/paper_files/ICML2011Socher_125.pdf.

GW Taylor, G E Hinton, and S Roweis. Modeling human motion using binary latent variables. *Advances in neural ...*, 2006. URL http://machinelearning.wustl.edu/mlpapers/paper_files/NIPS2006_693.pdf.

Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. *Proceedings of the 25th international conference on Machine learning - ICML '08*, pages 1096–1103, 2008. doi: 10.1145/1390156.1390294. URL http://portal.acm.org/citation.cfm?doid=1390156.1390294.

Bo Chen, G Polatkan, Guillermo Sapiro, David B. Dunson, and Lawrence Carin. The hierarchical beta process for convolutional factor analysis and deep learning. *... Machine Learning ...*, 2011. URL http://machinelearning.wustl.edu/mlpapers/paper_files/ICML2011Chen_251.pdf.

Li Fei-Fei, Rob Fergus, and Pietro Perona. One-shot learning of object categories. *IEEE transactions on pattern analysis and machine intelligence*, 28(4):594–611, April 2006. ISSN 0162-8828. doi: 10.1109/TPAMI.2006.79. URL http://www.ncbi.nlm.nih.gov/pubmed/16566508.

A Rodriguez, David B. Dunson, and Alan E. Gelfand. The nested Dirichlet process. *Journal of the American ...*, 2008. URL http://amstat.tandfonline.com/doi/full/10.1198/016214508000000553.

Evgeniy Bart, Ian Porteous, Pietro Perona, and Max Welling. Unsupervised learning of visual taxonomies. *2008 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8, June 2008. doi: 10.1109/CVPR.2008.4587620. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4587620.

KR Canini and TL Griffiths. Modeling human transfer learning with the hierarchical dirichlet process. *NIPS 2009 workshop: Nonparametric ...*, pages 1–3, 2009. URL http://www.gatsby.ucl.ac.uk/~dilan/abstractsNPB09/canini_abstract.pdf.

EB Sudderth, Antonio Torralba, William T. Freeman, and Alan S. Willsky. Describing visual scenes using transformed objects and parts. *International Journal of ...*, pages 1–53, 2008. URL http://link.springer.com/article/10.1007/s11263-007-0069-5.

Antonio Torralba, KP Murphy, and WT Freeman. Shared features for multiclass object detection. *... Category-Level Object Recognition*, 2006. URL http://link.springer.com/chapter/10.1007/11957959_18.

E. Bart and S. Ullman. Cross-Generalization: Learning Novel Classes from a Single Example by Feature Replacement. *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, 1:672–679. doi: 10.1109/CVPR.2005.117. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1467333.

Boris Babenko, Steve Branson, and Serge Belongie. Similarity metrics for categorization: from monolithic to category specific. *Computer Vision, 2009 ...*, pages 293–300, September 2009. doi: 10.1109/ICCV.2009.5459264. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5459264http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5459264.

Ruslan Salakhutdinov, JB Tenenbaum, and Antonio Torralba. Learning to Learn with Compound HD Models. *NIPS*, pages 1–9, 2011. URL http://books.nips.cc/papers/files/nips24/NIPS2011_1163.spotlight.pdf.

Mohammad Norouzi, Mani Ranjbar, and Greg Mori. Stacks of convolutional restricted Boltzmann machines for shift-invariant feature learning. *Computer Vision and Pattern ...*, 2009. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5206577.

Guillaume Desjardins and Yoshua Bengio. Empirical evaluation of convolutional RBMs for vision. *DIRO, Université de Montréal*, pages 1–13, 2008. URL http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:Empirical+Evaluation+of+Convolutional+RBMs+for+Vision#0.

GE Hinton. Training products of experts by minimizing contrastive divergence. *Neural computation*, 2002. URL http://www.mitpressjournals.org/doi/abs/10.1162/089976602760128018.

Honglak Lee, C Ekanadham, and A Ng. Sparse deep belief net model for visual area V2. *Advances in neural ...*, pages 1–8, 2007. URL http://machinelearning.wustl.edu/mlpapers/paper_files/NIPS2007_934.pdf.

Rakesh Chalasani and JC Principe. Deep Predictive Coding Networks. *arXiv preprint arXiv:1301.3541*, pages 1–13, 2013. URL http://arxiv.org/abs/1301.3541.

R P Rao and D H Ballard. Dynamic model of visual recognition predicts neural response properties in the visual cortex. *Neural computation*, 9(4):721–63, May 1997. ISSN 0899-7667. URL http://www.ncbi.nlm.nih.gov/pubmed/9161021.

Karl Friston. Hierarchical models in the brain. *PLoS computational biology*, 4 (11):e1000211, November 2008. ISSN 1553-7358. doi: 10.1371/journal.pcbi.1000211. URL http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=2570625&tool=pmcentrez&rendertype=abstract.

Youngmin Cho. Kernel methods for deep learning. pages 1–9, 2012. URL http://cseweb.ucsd.edu/~yoc002/paper/thesis_youngmincho.pdf.

B Schölkopf, Alexander Smola, and KR Müller. Nonlinear component analysis as a kernel eigenvalue problem. *Neural computation*, (44), 1998.

Florian Yger, Maxime Berar, Gilles Gasso, and Alain Rakotomamonjy. A supervised strategy for deep kernel machine. *ESANN 2011 ...*, (April):27–29, 2011. URL http://hal.archives-ouvertes.fr/hal-00668302/.

Roman Rosipal, LJ Trejo, and Bryan Matthews. Kernel PLS-SVC for linear and nonlinear classification. *ICML*, 2003. URL http://www.aaai.org/Papers/ICML/2003/ICML03-084.pdf.

P Hamel, S Lemieux, Y Bengio, and Douglas Eck. Temporal Pooling and Multiscale Learning for Automatic Annotation and Ranking of Music Audio. *ISMIR*, 2011.

S. Renals and N. Morgan. Connectionist probability estimators in HMM speech recognition. *Speech and Audio ...*, 2(1):161–174, 1994. ISSN 10636676.

doi: 10.1109/89.260359. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=260359http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=260359.

Abdel-rahman Mohamed, Dong Yu, and Li Deng. Investigation of Full-Sequence Training of Deep Belief Networks for Speech Recognition. (September):2846–2849, 2010.

Dong Yu, Li Deng, and G Dahl. Roles of pre-training and fine-tuning in context-dependent DBN-HMMs for real-world speech recognition. *Proc. NIPS Workshop on Deep ...*, 2010a. URL http://research.microsoft.com/pubs/143619/dbn4asr-nips2010.pdf.

Frank Seide, Gang Li, and Dong Yu. Conversational Speech Transcription Using Context-Dependent Deep Neural Networks. *INTERSPEECH*, (August):437–440, 2011a. URL http://research.microsoft.com/pubs/153169/CD-DNN-HMM-SWB-Interspeech2011-Pub.pdf.

Frank Seide, Gang Li, Xie Chen, and Dong Yu. Feature engineering in Context-Dependent Deep Neural Networks for conversational speech transcription. *2011 IEEE Workshop on Automatic Speech Recognition & Understanding*, pages 24–29, December 2011b. doi: 10.1109/ASRU.2011.6163899. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6163899.

Dong Yu, Frank Seide, Gang Li, and Li Deng. Exploiting sparseness in deep neural networks for large vocabulary speech recognition. *..., Speech and Signal Processing (...*, pages 4409–4412, March 2012a. doi: 10.1109/ICASSP.2012.6288897. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6288897http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6288897.

Geoffrey Hinton, Li Deng, Dong Yu, George Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Vincent Vanhoucke, Patrick Nguyen, Tara Sainath, and Brian Kingsbury. Deep Neural Networks for Acoustic Modeling in Speech Recognition. *IEEE Signal Processing Magazine*, 2(November):1–27, 2012.

Dong Yu, Xin Chen, and Li Deng. Factorized deep neural networks for adaptive speech recognition. *... Machine Learning for Speech ...*, 2012b. URL http://research.microsoft.com/pubs/162504/f-DNN.pdf.

Abdel-rahman Mohamed, George Dahl, and Geoffrey Hinton. Deep Belief Networks for phone recognition. *Science*, 4(5):1–9, 2009. ISSN 19416016. doi: 10.4249/scholarpedia.5947. URL http://www.cs.utoronto.ca/~gdahl/papers/dbnPhoneRec.pdf.

GW Taylor. *Composable, distributed-state models for high-dimensional time series*. 2009. URL http://dl.acm.org/citation.cfm?id=1925632.

Ilya Sutskever and Geoffrey Hinton. Learning multilevel distributed representations for high-dimensional sequences. *International Conference on ...*, 2007. URL http://machinelearning.wustl.edu/mlpapers/paper_files/AISTATS07_SutskeverH.pdf.

Ilya Sutskever, G Hinton, and G Taylor. The recurrent temporal restricted boltzmann machine. *Advances in Neural ...*, 2008. URL http://machinelearning.wustl.edu/mlpapers/paper_files/NIPS2008_0583.pdf.

Ivan Titov and James Henderson. Fast and Robust Multilingual Dependency Parsing with a Generative Latent Variable Model. *EMNLP-CoNLL*, 2007. URL http://acl.ldc.upenn.edu/D/D07/D07-1099.pdf.

Itamar Arel, Derek Rose, and Tom Karnowski. A deep learning architecture comprising homogeneous cortical circuits for scalable spatiotemporal pattern inference. *... Workshop on Deep Learning ...*, 2009b. URL http://research.microsoft.com/en-us/um/people/dongyu/nips2009/papers/Arel-DeSTIN_NIPStpk2.pdf.

HB Barlow. Unsupervised learning. *Neural computation*, 1989. URL http://www.mitpressjournals.org/doi/abs/10.1162/neco.1989.1.3.295.

TP Karnowski. Deep Machine Learning with Spatio-Temporal Inference. 2012. URL http://trace.tennessee.edu/utk_graddiss/1315/.

Hossein Mobahi, R Collobert, and Jason Weston. Deep learning from temporal coherence in video. *... Conference on Machine Learning*, 2009. URL http://dl.acm.org/citation.cfm?id=1553469.

AJ Lockett and R Miikkulainen. Temporal convolution machines for sequence learning. *To Appear*, pages 1–8, 2009. URL ftp://www.cs.utexas.edu/pub/neural-nets/papers/lockett-tcm.pdf.

EH Adelson, CH Anderson, and JR Bergen. Pyramid methods in image processing. *RCA ...*, 1984. URL http://w3.cs.huji.ac.il/course/2003/impr/lectures2001/pyramid.pdf.

M Bister, J Cornelis, and A Rosenfeld. A critical view of pyramid segmentation algorithms. *Pattern Recognition Letters*, 11(September):605–617, 1990. URL http://www.sciencedirect.com/science/article/pii/016786559090013R.

PJ Burt, TH Hong, and A Rosenfeld. Segmentation and estimation of image region properties through cooperative hierarchial computation. *Systems, Man and ...*, 75(12): 802–809, 1981. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4308619.

John Lafferty, A McCallum, and FCN Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. 2001(Icml):282–289, 2001. URL http://repository.upenn.edu/cis_papers/159/.

H. Hermansky and S. Sharma. Temporal patterns (TRAPs) in ASR of noisy speech. *1999 IEEE International Conference on Acoustics, Speech, and Signal Processing. Proceedings. ICASSP99 (Cat. No.99CH36258)*, pages 289–292 vol.1, 1999. doi: 10.1109/ICASSP.1999.758119. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=758119.

Asela Gunawardana, Milind Mahajan, Alex Acero, and JC Platt. Hidden conditional random fields for phone classification. *INTERSPEECH*, 2005. URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.113.8190&rep=rep1&type=pdf.

Yun-Hsuan Sung and Dan Jurafsky. Hidden Conditional Random Fields for phone recognition. *2009 IEEE Workshop on Automatic Speech Recognition & Understanding*, pages 107–112, December 2009. doi: 10.1109/ASRU.2009.5373329. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5373329.

Dong Yu, Shizhen Wang, and Li Deng. Sequential labeling using deep-structured conditional random fields. *. . . in Signal Processing, IEEE Journal of*, 4(6):965–973, 2010b. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5570918.

Dong Yu and Li Deng. Deep-structured hidden conditional random fields for phonetic recognition. *INTERSPEECH*, (September):2986–2989, 2010. URL http://131.107.65.14/pubs/135407/deepCRF-interspeech2010.pdf.

G Andrew and J Bilmes. Sequential deep belief networks. *Acoustics, Speech and Signal Processing . . .*, pages 4265–4268, 2012. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6288861.

Galen Andrew. New Techniques for Modeling Sequential Data. *homes.cs.washington.edu*, pages 1–26, 2009. URL http://homes.cs.washington.edu/~galen/files/quals.pdf.

JL Elman. Finding structure in time. *Cognitive science*, 14(2):179–211, June 1990. ISSN 03640213. doi: 10.1016/0364-0213(90)90002-E. URL http://doi.wiley.com/10.1016/0364-0213(90)90002-Ehttp://www.sciencedirect.com/science/article/pii/036402139090002E.

Michael I. Jordan. Attractor dynamics and parallelism in a connectionist sequential machine. In *Proceedings of the Eighth Annual Conference of the Cognitive*

Science Society, pages 531–546. IEEE, 1986. ISBN 0-8186-2015-3. URL http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:Attractor+dynamics+and+parallelism+in+a+connectionist+sequential+machine#0.

KJ Lang, AH Waibel, and GE Hinton. A time-delay neural network architecture for isolated word recognition. *Neural networks*, 3:23–43, 1990. URL http://www.sciencedirect.com/science/article/pii/089360809090044L.

Herbert Jaeger. The" echo state" approach to analysing and training recurrent neural networks-with an erratum note'. *Bonn, Germany: German National Research ...*, pages 1–47, 2001. URL http://minds.jacobs-university.de/sites/default/files/uploads/papers/EchoStatesTechRep.pdf.

I Sutskever, J Martens, and G Hinton. Generating text with recurrent neural networks. *Proceedings of the ...*, 2011. URL http://machinelearning.wustl.edu/mlpapers/paper_files/ICML2011Sutskever_524.pdf.

Alex Graves. *Supervised sequence labelling with recurrent neural networks*. 2012. ISBN 9783642212703. URL http://books.google.com/books?hl=en&lr=&id=4UauNDGQWN4C&oi=fnd&pg=PR1&dq=Supervised+Sequence+Labelling+wit+Recurrent+Neural+Networks&ots=IIHH-W-zr7&sig=0VCEzEvfKWQFjPfOSeYvM8cpFJg.

Michael Schuster. On supervised learning from sequential data with applications for speech recognition. *Daktaro disertacija, Nara Institute of Science and ...*, 1999. URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.29.7060&rep=rep1&type=pdf.

Toshiaki Fukada, Mike Schuster, and Yoshinori Sagisaka. Phoneme Boundary Estimation Using Bidirectional Recurrent Neural Networks and its Applications. *Systems and Computers in Japan*, 30(4), 1999.

Jinmiao Chen and NS Chaudhari. Capturing long-term dependencies for protein secondary structure prediction. *Advances in Neural Networks-ISNN 2004*, pages 1–6, 2004. URL http://link.springer.com/chapter/10.1007/978-3-540-28648-6_79.

Pierre Baldi, Sø ren Brunak, Paolo Frasconi, Gianluca Pollastri, and Giovanni Soda. Bidirectional dynamics for protein secondary structure prediction. *Sequence Learning*, pages 80–104, 2001. URL http://link.springer.com/chapter/10.1007/3-540-44565-X_5.

M. Schuster and K.K. Paliwal. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681, 1997. ISSN 1053587X. doi:

10.1109/78.650093. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=650093.

P Baldi, S Brunak, P Frasconi, G Soda, and G Pollastri. Exploiting the past and the future in protein secondary structure prediction. *Bioinformatics (Oxford, England)*, 15(11):937–46, November 1999. ISSN 1367-4803. URL http://www.ncbi.nlm.nih.gov/pubmed/10743560.

Oriol Vinyals, SV Ravuri, and Daniel Povey. Revisiting recurrent neural networks for robust ASR. *Acoustics, Speech and Signal ...*, pages 4085–4088, 2012. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6288816.

S Hochreiter, Y Bengio, P Frasconi, and J Schmidhuber. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies. 2001. URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.24.7321&rep=rep1&type=pdf.

Y Bengio, P Simard, and P Frasconi. Learning long-term dependencies with gradient descent is difficult., January 1994. ISSN 1045-9227. URL http://www.ncbi.nlm.nih.gov/pubmed/18267787.

Daan Wierstra, Faustino J. Gomez, and Jürgen Schmidhuber. Modeling systems with internal state using evolino. *Proceedings of the 2005 conference on Genetic and evolutionary computation - GECCO '05*, page 1795, 2005. doi: 10.1145/1068009.1068315. URL http://portal.acm.org/citation.cfm?doid=1068009.1068315.

Jürgen Schmidhuber, Daan Wierstra, Matteo Gagliolo, and Faustino Gomez. Training recurrent networks by Evolino. *Neural computation*, 19(3):757–79, March 2007. ISSN 0899-7667. doi: 10.1162/neco.2007.19.3.757. URL http://www.ncbi.nlm.nih.gov/pubmed/17298232.

Tsungnan Lin, BG Horne, P Tino, and CL Giles. Learning long-term dependencies in NARX recurrent neural networks. *Neural Networks, IEEE ...*, I(6):1329–1338, 1996. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=548162.

T. A. Plate. Holographic Recurrent Networks. *Citeseer*, 5(1):1–8. URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.50.4046&rep=rep1&type=pdf.

MC Mozer. Induction of multiscale temporal structure. *NIPS*, 1991. URL http://pdf.aminer.org/000/516/394/induction_of_multiscale_temporal_structure.pdf.

J Schmidhuber. Learning complex, extended sequences using the principle of history compression. *Neural Computation*, 3254:2–8, 1992. URL http://www.mitpressjournals.org/doi/abs/10.1162/neco.1992.4.2.234.

S Hochreiter and J Schmidhuber. Long short-term memory. *Neural computation*, 1997. URL http://www.mitpressjournals.org/doi/abs/10.1162/neco.1997.9.8.1735.

F a Gers, J Schmidhuber, and F Cummins. Learning to forget: continual prediction with LSTM. *Neural computation*, 12(10):2451–71, October 2000. ISSN 0899-7667. URL http://www.ncbi.nlm.nih.gov/pubmed/11032042.

FA Gers, NN Schraudolph, and J Schmidhuber. Learning precise timing with LSTM recurrent networks. *... of Machine Learning ...*, 3:115–143, 2003. URL http://dl.acm.org/citation.cfm?id=944925.

Felix Gers. Long Short-Term Memory in Recurrent Neural Networks. *Lausanne, EPFL*, 2366, 2001. URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.28.6677&rep=rep1&type=pdf.

FA Gers and E Schmidhuber. LSTM recurrent networks learn simple context-free and context-sensitive languages. *... Networks, IEEE Transactions on*, 2001. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=963769.

Sepp Hochreiter, Martin Heusel, and Klaus Obermayer. Fast model-based protein homology detection without alignment. *Bioinformatics (Oxford, England)*, 23(14):1728–36, July 2007. ISSN 1367-4811. doi: 10.1093/bioinformatics/btm247. URL http://www.ncbi.nlm.nih.gov/pubmed/17488755.

D. Eck and J. Schmidhuber. Finding temporal structure in music: blues improvisation with LSTM recurrent networks. *Proceedings of the 12th IEEE Workshop on Neural Networks for Signal Processing*, pages 747–756. doi: 10.1109/NNSP.2002.1030094. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1030094.

Bram Bakker. Reinforcement Learning with Long Short-Term Memory. *NIPS*, 2001. URL http://www-2.cs.cmu.edu/Groups/NIPS/NIPS2001/papers/psgz/CN05.ps.gz.

Alex Graves and Jürgen Schmidhuber. Framewise phoneme classification with bidirectional LSTM and other neural network architectures. *Neural networks : the official journal of the International Neural Network Society*, 18(5-6):602–10, 2004. ISSN 0893-6080. doi: 10.1016/j.neunet.2005.06.042. URL http://www.ncbi.nlm.nih.gov/pubmed/16112549.

Alex Graves and S Fernández. Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. *Proceedings of the 23rd ...*, 2006. URL http://dl.acm.org/citation.cfm?id=1143891.

Marcus Liwicki, A Graves, H Bunke, and J Schmidhuber. A novel approach to on-line handwriting recognition based on bidirectional long short-term memory networks. *... Analysis and Recognition*, 2007. URL http://www.dfki.de/~liwicki/pdf/LiGrBuSch07-01.pdf.

Alex Graves and Marcus Liwicki. Unconstrained on-line handwriting recognition with recurrent neural networks. *... in Neural ...*, pages 1–8, 2007. URL http://machinelearning.wustl.edu/mlpapers/paper_files/NIPS2007_894.pdf.

A. J. Robinson and F Fallside. Static and Dynamic Error Propagation Networks with Application to Speech Coding. *Proceedings of Neural Information Processing Systems, American Institute of Physics*, 1987.

Ronald J Williams and David Zipser. Gradient-Based Learning Algorithms for Recurrent Networks and Their Computational Complexity. pages 1–45, 1995.

Justin Bayer and Daan Wierstra. Evolving memory cell structures for sequence learning. *Artificial Neural Networks ...*, pages 1–10, 2009. URL http://link.springer.com/chapter/10.1007/978-3-642-04277-5_76.

a. Graves and J. Schmidhuber. Framewise phoneme classification with bidirectional LSTM networks. *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, 4:2047–2052. doi: 10.1109/IJCNN.2005.1556215. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1556215.

J. Chen and N. S. Chaudhari. Protein Secondary Structure Prediction with bidirectional LSTM networks. In *Post-Conference Workshop on Computational Intelligence Approaches for the Analysis of Bio-data (CI-BIO)*, Montreal, Canada, 2005.

Trias Thireou and Martin Reczko. Bidirectional Long Short-Term Memory Networks for predicting the subcellular localization of eukaryotic proteins. *IEEE/ACM transactions on computational biology and bioinformatics / IEEE, ACM*, 4(3):441–6, 2007. ISSN 1545-5963. doi: 10.1109/tcbb.2007.1015. URL http://www.ncbi.nlm.nih.gov/pubmed/17666763.

a J Robinson. An application of recurrent nets to phone probability estimation. *IEEE transactions on neural networks / a publication of the IEEE Neural Networks Council*, 5(2):298–305, January 1994. ISSN 1045-9227. doi: 10.1109/72.279192. URL http://www.ncbi.nlm.nih.gov/pubmed/18267798.

J Neto, L Almeida, M Hochberg, and C Martins. Speaker-adaptation for hybrid HMM-ANN continuous speech recognition system. (September):2171–2174, 1995. URL http://www.era.lib.ed.ac.uk/handle/1842/1274.

DJ Kershaw, MM Hochberg, and AJ Robinson. *Context-dependent classes in a hybrid recurrent network-HMM speech recognition system*, volume 0. 1995. URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.49.8984&rep=rep1&type=pdf.

A Senior and T Robinson. Forward-backward retraining of recurrent neural networks. *... in Neural Information Processing Systems*, 3:3–9, 1996. URL ftp://mi.eng.cam.ac.uk/pub/pub/reports/auto-pdf/senior_fbrnn.pdf.

AJ Robinson, L Almeida, JM Boite, H Bourlard, F Fallside, M Hochberg, D Kershaw, P Kohn, Y Konig, N Morgan, JP Neto, S Renals, M Saerens, and C Wooters. A neural network based, speaker independent, large vocabulary, continuous speech recognition system: The WERNICKE project. *To appear in Proc. ...*, 1993. URL ftp://svr-www.eng.cam.ac.uk/reports/auto-pdf/wernicke_eurospeech93.pdf.

Jeff Hawkins and Sandra Blakeslee. *On intelligence*. 2004. URL http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:On+Intelligence#2http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:On+intelligence#2.

Itamar Arel, DC Rose, and TP Karnowski. Deep machine learning:A new frontier in artificial intelligence research. *Computational Intelligence ...*, (November):13–18, 2010. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5605630.

Davide Maltoni. Pattern Recognition by Hierarchical Temporal Memory. Technical report, 2011.

Numenta. Hierarchical Temporal Memory including HTM Cortical Learning Algorithms. Technical report, 2011.

D George. *How the brain might work: A hierarchical and temporal model for learning and recognition*. PhD thesis, 2008. URL http://alpha.tmit.bme.hu/speech/docs/education/02_DileepThesis.pdf.

J Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. 1988. URL http://books.google.com/books?hl=en&lr=&id=AvNID7LyMusC&oi=fnd&pg=PA1&dq=Probabilistic+Reasoning+in+Intelligent+Systems:+Networks+of+Plausible+Inference&ots=FY1MQipz-9&sig=fLiQDZzYdP7ntyU9bEQ3B9xV-j0.

JW Miller and PH Lommel. Biomimetic sensory abstraction using hierarchical quilted self-organizing maps. *Optics East 2006*, 1(617), 2006. URL http://proceedings.spiedigitallibrary.org/proceeding.aspx?articleid=1332574.

Sven Behnke. *Hierarchical neural networks for image interpretation*, volume 2766 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003. ISBN 978-3-540-40722-5. doi: 10.1007/b11963. URL http://www.springerlink.com/index/10.1007/b11963http://books.google.com/books?hl=en&lr=&id=nj8z5NuG2SUC&oi=fnd&pg=PR5&dq=Hierarchical+Neural+Networks+for+Image+Interpretation&ots=fjN9I1uG8c&sig=GPrlcPvc7hn5A9Irm0x2oNnWZzI.

Numenta Inc. Numenta — NuPIC. URL http://numenta.org/.

Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. *arXiv preprint arXiv:1303.5778*, (3), 2013. URL http://arxiv.org/abs/1303.5778.