Deep Learning : Towards Building Truly Intelligent Machines

Saurabh V. Pendse
Department of Computer Science
North Carolina State University
Raleigh, North Carolina 27695-8206
Email: svpendse@ncsu.edu

I. Abstract

Abstract—**This paper provides an introduction to the field of deep learning and the associated frameworks such as Restricted Boltzmann Machines (RBMs), Deep Belief Networks (DBNs), Convolutional Neural Networks (CNNs), Autoencoders and others. We first give a short background of machine learning and shallow architectures like Perceptrons, SVMs and provide the motivations behind deep learning. We then proceed to discuss some important breakthroughs in training deep architectures as well as examine the state-of-the-art deep learning algorithms. We illustrate the usefulness of deep learning algorithms by discussing real-world applications and highlight the challenges, trends and future work in the field of deep learning.**

Keywords-**Machine Learning, Intelligence, Cognition, Hierarchy, Neural Networks, Pre-training**

II. Introduction

A fundamental goal behind half a century of AI research has been to enable machines to model the world well enough to exhibit what we define as "intelligence". To achieve this, it is clear that a large quantity of information about our world should somehow be stored, explicitly or implicitly, in machines. The vast body of research on building intelligent systems indicates that the problem of AI seems to be a problem of knowledge representation. According to Jeff Hawkins, "Finding a good representation for the massive amount of knowledge about the world is hard enough, it is compounded by the need to efficiently extract contextually relevant knowledge depending on the situation" [14].

The traditional approach to AI was born with the digital computer. Alan Turing was one of the proponents of the idea of a general-purpose computer. In a seminal paper [40], he described the idea of a universal turing machine with three essential parts : a processing box, a paper tape, and a device that reads and writes marks on the tape as it moves back and forth. This became the basis for all of computing today i.e. the CPU and the concept of linear memory in the computer.

Today, computers based on this fundamental principle can be used to emulate a lot of things. Although much progress has been made in understanding and improving learning algorithms, the challenge of AI still remains: Do we have algorithms that can infer enough semantic concepts in order to be able to interact with most humans using these contexts? The answer is *"no"*. If we consider image recognition, one of the best specified of the AI problems, we realize that we do not yet have learning algorithms that can discover the many visual and semantic concepts that would seem trivial for humans. The situation is similar for other problems as well.

The human brain is undoubtedly one of the most complicated systems in the world. It consists of billions of cells, known as neurons, with trillions of connections between them. After almost a century of neuroscience research, scientists still have no definitive idea about how the brain is able to perform tasks so well. Brains are known to have a completely different architecture as compared to machines. There has been a significant amount of progress in understanding the structure of the brain in the past two decades [7], [13], [44].

Deep learning is a subfield of Artificial Intelligence that has recently gained a lot of prominence [3], [5], [17]–[19]. It is based on modeling the structure of the human brain, which is capable of processing complex input data, quickly producing thought patterns from the collected knowledge about the world, and solving different kinds of complicated learning and cognition problems well. It aims to switch these features of the human brain into a learning model which can deal with high-dimensional data, support fast learning algorithms and perform well in the complicated AI problems such as Computer Vision or Natural Language Processing. Deep architectures are a promising step towards creating such a model.

In this paper, we present a review of the fundamental ideas and motivation behind deep learning and examine the state-of-the-art deep learning frameworks and computing models. Section III starts with the motivating factors behind deep learning. Section IV explores the history of learning architectures as well as the current state-of-the-art deep learning algorithms. Section V discusses some of the recent and interesting applications of these algorithms in various domains. Section VI discusses the pressing challenges, trends and future work in deep learning. Finally, Section VII provides the overall conclusions of this paper.

III. Motivation

The major motivations for studying deep learning architectures and algorithms can be described as follows :

A. *Depth and Performance*

There exists a strong relationship between a machine learning architecture's depth and its performance in real-world problems. For most cases like Logic Gates, Radial Basis Functions (RBF) units like in SVM, a depth of 2 is sufficient to represent any function with a given target accuracy. However, this comes at a price of increased complexity. Theoretical results indicate that there exist function classes for which the complexity grows exponentially with the input size. Deep architectures can be viewed as a type of factorization. There exist many functions that can be represented efficiently with a deep architecture, but cannot be represented efficiently with a shallow architecture [3], [10]. This is indicative of some high-level patterns in the underlying function to be represented.

B. *Architecture of the brain*

The brain has a deep architecture. From an AI perspective, the part of the brain that is most interesting is the neocortex (60% of the volume of the brain). It is the location of all high-level knowledge. It is a flat sheet of cells, roughly equal to the size of a table cloth and consists of about 30 billion cells (or neurons). Neuroscientists have recently discovered that the neocortex has a remarkably regular structure. Knowledge arises from connections between neurons in different regions of the neocortex which has a hierarchical self-learning structure. For example, the well-studied visual cortex shows a hierarchy of layers of neurons, each of which contains a representation of the input. Each level of this feature hierarchy represents the input at a different level of abstraction, with better abstractions further up in the hierarchy, defined as compositions of lower-level features.

C. *Nature of Cognitive processes*

It has been scientifically proven [7], [13], [44] that humans organize their ideas and concepts hierarchically. We first learn simpler concepts and then compose them to represent more abstract ones. For example, large-scale software systems are often developed using a modular approach, with each module performing some simple transformation of data, and the compositions of modules ultimately resulting in complex and capable systems.

The brain learns a generative model of the given input across multiple layers of features in the neocortex without any supervision. Traditional algorithms like backpropagation, SVMs require labeled training data. Moreover, the learning time does not scale well in networks with multiple hidden layers. Contrary to this, the brain uses almost all unlabeled data, and fits about 10^{14} connection weights in time of the order of 10^9 seconds (average human lifespan). Labels alone cannot possibly provide enough information to achieve this kind of learning.

IV. Deep Architectures

The computations performed by a learnt function can be decomposed into a flow graph of simpler operations. A flow graph is nothing but a graphical representation of a computation, wherein each node represents an atomic computation and the result of the computation is applied to the values at the children of that node. The Figure 1 illustrates an example.

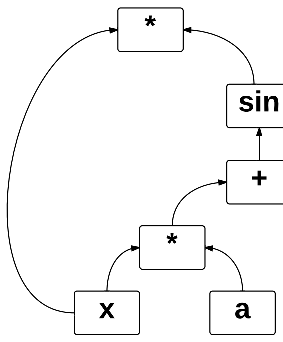

Fig. 1. A flow graph for computing the function $x \cdot \sin(ax+b)$ [2].

A particular property of such flow graphs is depth i.e. the length of the longest path from an input to an output. Traditional feedforward neural networks can be considered to have depth equal to the number of layers. Support Vector Machines have a depth of 2, one for the kernel outputs or for the feature space, and one for the linear combination producing the output [3]. A complex learning problem, when solved on an architecture with insufficient depth may require an exponential-sized architecture. However, the same problem with sufficient depth results in a compact representation [3]. Although very promising, training deep architectures was extremely difficult which resulted in their failure. However, a breakthrough [19] in 2006 unlocked the true potential of deep architectures and has since resulted in widespread success.

In contrast to shallow architectures like perceptrons, SVMs or kernel machines which only contain a fixed base function and typically a linear weight-combination layer, deep architectures refer to multi-layer networks where two adjacent layers are connected to each other in a specific way. Bengio and Lecun [5] state that "deep architectures are compositions of many layers of adaptive non-linear components, in other words; they are cascades of parameterized non-linear modules that contain trainable parameters at all levels."

We will now understand how deep architectures relate to two core problems in Artificial Intelligence : Knowledge Representation and Learning, by providing a brief history about the developments of deep and shallow networks.

A. *Historical Development of Learning Architectures*

Shallow architectures have been around since the very inception of AI. There is an interesting history behind the changes in attitude of the AI community towards deep and shallow architectures.

1) Perceptrons: Frank Rosenblatt proposed the idea of an artificial brain, or the Perceptron, around 1960 [35]. It had one hand-crafted feature layer, and tried to implement object recognition by learning weight vectors combining all the features in a particular manner. Its capability to classify basic shapes like triangles and squares led researchers to believe that a machine capable of sensing, learning and recognizing like humans could be invented. However, one of the fundamental limitations of the Perceptron was that the feature layer was fixed and crafted by humans which contradicts to the notion of ”intelligence”. In 1969, Minsky and Papert [29] demonstrated that its single layer structure limited the functions it could learn. For instance, a XOR function could not be learnt.

2) Neural Networks with hidden layers: Perceptrons provided an impetus to neural network research, although they were limited in their capabilities. In 1985, Geoffery Hinton [1], [39] replaced the original fixed feature layer with several hidden layers, creating the second generation neural network. Such a network could learn more complicated functions compared to Perceptrons, using a well-known learning algorithm known as Backpropagation [36]. It worked by back-propagating the error signal computed at the output layer to get derivatives for learning, in order to update the weights of the network until convergence was reached. However it lacked the ability to train using unlabeled data, which was relevant to most real-world problems. Moreover, the correcting signal could be weakened when it passed back via multiple layers. The training process became very slow and infeasible across multiple hidden layers and the resulting solutions were often local optima, rather than the global optima.

While some researchers made improvements to Hinton’s neural networks, others made improvements on the original Perceptrons, creating a family of algorithms called Support Vector Machines (SVMs). These attracted most researchers’ attention, which thwarted the developments of the neural network.

3) Support Vector Machines (SVMs): The concept of Support Vector Machines was first proposed by Vapnik in 1995 [24]. It is based on a statistical learning theory, that turns the hand-crafted feature layer in the original Perceptrons into a feature layer generated using a fixed method. This method is known as the kernel trick, which maps the input data into a high-dimensional space. Then, a clever optimization technique is adopted to learn the weights combining the feature and data corresponding to the output [31].

SVMs make learning fast and easy, due to their simple structure. SVMs work well for many AI problems (e.g. pattern recognition) wherein the data has a relatively simple structure i.e. has a small number of features or which doesn’t contain hierarchical structures [31]. However, when the data itself contains complicated features, SVMs tend to perform worse due to their simple structure [19], [42]. Even if the kernel function maps the input data to a more complicated high-dimensional space, the procedure is still ”static” or ”fixed”. Since the kernel functions use a pre-determined mapping method for every data, SVMs might not be able to extract all the information present in the data.

One way to mitigate this problem is to add prior knowledge to the SVM model in order to obtain a better feature layer [24]. However, this involves human intervention and is highly dependent on the prior knowledge added, which again diverges from the goal of building truly intelligent machines capable of autonomous learning. SVMs are still a kind of Perceptrons which use kernel functions instead of hand-crafted features, an optimization technique instead of the original Backpropagation algorithm and can deal with unlabeled data. Despite the fact that SVMs are good at solving many AI problems, they are not a good trend for AI due a fundamental caveat : their performance is highly dependent on the choice of the appropriate kernel function.

In order to build truly intelligent machines, an architecture should be capable of learning features autonomously from the input data, as well as dealing with unlabeled data. Moreover, the training algorithms should be efficient and general and it must be applicable to a variety of AI problems. With these goals in mind, some researchers started looking back at Hinton’s multi-layer neural networks, trying to exploit its advantages and overcome its limitations.

B. Breakthrough in Learning Deep Architectures

Before 2006, all attempts at training deep architectures failed miserably. Training deep supervised feedforward neural networks in the same manner as shallow architectures (1 or 2 hidden layers) yielded worse results both in terms of training and test error.

However, a new class of layer-by-layer pre-training algorithms [4], [18], [19], [28] proposed in 2006 and onwards completely changed the scenario. These algorithms were based on the following key principles :

- Each layer in the deep network was (pre-)trained in a completely unsupervised manner.
- The layers were stacked one on top of the other. The output representation at each layer was the input for the next layer.
- Supervised training was used on the entire network to fine-tune all the layers.

Since then, a plethora of papers on the subject of deep learning have been published, with many of them using other principles to guide training of intermediate representations.

C. Restricted Boltzmann Machines & Deep Belief Networks

Restricted Boltzmann Machines (RBMs) are energy-based models that associate a scalar energy to each configuration of the variables of interest. Learning corresponds to modifying the energy function so that its shape has desirable properties. For example, desirable configurations would likely have a low energy. Such models define a probability distribution through an energy function $E(x)$, such that

$$p(x) = \frac{e^{-E(x)}}{N} \tag{1}$$

N is the normalizing factor, or the *partition function* given by :

$$N = \sum_{x} e^{-E(x)}$$

Such models can be learnt by performing stochastic gradient descent on the empirical negative log-likelihood of the training data. In case of RBMs, the energy function is linear in its free parameters. To make them powerful enough to represent complex distributions, hidden variables are introduced. Moreover, the connections are restricted to those without visible-visible and hidden-hidden connections. The typical configuration of a RBM also uses binary units (where v_i and $h_j \in \{0,1\}$). The structure of a RBM is shown in the Figure 2.

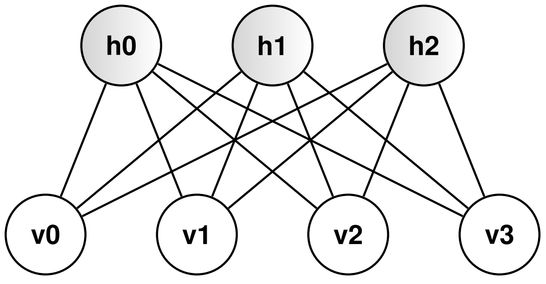

Fig. 2. The structure of a Restricted Boltzmann Machine [19]. It consists of visible and hidden units without any lateral connections between them.

A joint configuration of the visible and hidden units (v,h) has the energy given by [19] :

$$E(v,h) = -\sum_{i\in\text{visible}} a_i v_i - \sum_{j\in\text{hidden}} b_j h_j - \sum_{i,j} v_i h_j w_{ij} \tag{2}$$

where v_i, h_j are the binary states of the visible unit i and the hidden unit j, a_i, b_j are their biases, and w_{ij} represents the weight between them. Taking the negative of the derivative with respect to w_{ij}, we have :

$$-\frac{\partial E(v,h)}{\partial w_{ij}} = v_i h_j \tag{3}$$

In terms of input probability distribution (Equation 1), we have :

$$\frac{\partial \log p(v)}{\partial w_{ij}} = < v_i h_j >^0 - < v_i h_j >^\infty \tag{4}$$

where $< \ldots >$ denotes the expectation of a random variable, $< v_i h_j >^0$ is the positive gradient and $< v_i h_j >^\infty$ is the negative gradient. These gradients are calculated using alternative Gibbs sampling, which consists of running a Markov chain of convergence, using Gibbs sampling as the transition operator. The goal is to minimize the error between the input and the model's generative expectation of the input. Furthermore, Contrastive Divergence can be used to speed up the learning process [17], [19]. This requires execution of only one step to get $< v_i h_j >^1$. The weight updates are given by :

$$\triangle w_{ij} = < v_i h_j >^0 - < v_i h_j >^1 \tag{5}$$

Since RBMs do not have any lateral connections, the visible and hidden units are conditionally independent given one another. Thus,

$$p(h|v) = \prod_i p(h_i|v) \tag{6}$$

$$p(v|h) = \prod_j p(v_j|h) \tag{7}$$

Deep Belief Networks (DBNs) are graphical models which learn to extract a deep hierarchical representation of the input data, formed by stacking RBMs on top of each other. They model a joint distribution between an input x and the m hidden layers as follows :

$$P(x,h^1,h^2,\ldots,h^m) = \left(\prod_{k=0}^{m-2} P(h^k|h^{k+1})\right) P(h^{m-1},h^m) \tag{8}$$

where $P(h^{k-1}|h^k)$ is a conditional distribution for the visible units of the RBM at layer k, and $P(h^{m-1},h^m)$ is the visible-hidden joint distribution at the top-layer RBM. The stacking is shown in the Figure 3.

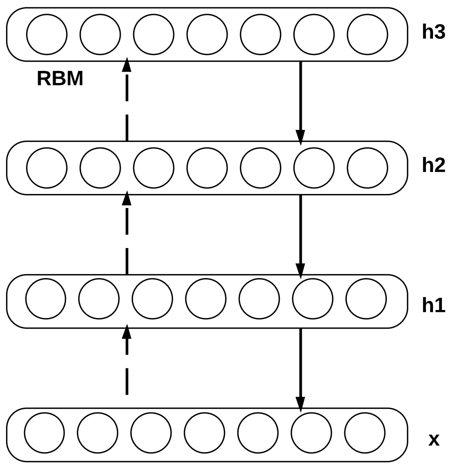

Fig. 3. The stacking of RBMs to form a Deep Belief Network

The greatest advantage of DBNs is their capability of learning features, which is achieved by an unsupervised greedy layer-by-layer training strategy (known as pre-training) [19] where the higher-level features are learnt from the lower layers and are expected to better capture the information contained in the data. After the pre-training, the weights between adjacent layers have values that encode the information contained in the data.

To further improve the performance, these weights are altered using a supervised fine-tuning process. If the pre-trained DBN is used as a discriminative model, Backpropagation is used to adjust the detection weights by supervised learning using the labeled data. It is necessary to ensure that the learning rate is set to an appropriate value; too large a value will greatly influence the pre-trained weights while too small a value will lead to a slow training process. If the pre-trained DBN is used as a generative model, a wake-sleep algorithm [16] is used to tune the DBN weights.

D. Autoencoders

Autoencoders [3] are another class of deep architectures which are similar in their functional form to RBMs. However, their interpretation and procedures used for training them are quite different. Deterministic autoencoders consider the real valued mean as their hidden representation while RBMs sample a binary hidden representation from the mean. However, after the initial pre-training, the layers of a RBM are typically used by propagating the real-valued means through them.

Consider a $d-$dimensional input vector $\mathbf{x}$ which is transformed into its d'-dimensional hidden representation $\mathbf{y}$ using a deterministic mapping function i.e. the encoder. A typical form of the mapping function is given by :

$$\mathbf{y} = f(\mathbf{x}) = s(\mathbf{W}\mathbf{x} + \mathbf{b}) \tag{9}$$

where $\mathbf{W}$ is the $d \times d'$ weight matrix, while $\mathbf{b}$ is the $d' \times 1$ bias vector. The resulting representation $\mathbf{y}$ is then mapped back to a reconstructed d-dimensional vector $\mathbf{z}$ (also known as the pre-image of $\mathbf{x}$ using a mapping function g i.e. the decoder).

$$\mathbf{z} = g(\mathbf{y}) = s(\mathbf{W}'\mathbf{y} + \mathbf{b}') \tag{10}$$

It should be noted that $\mathbf{z}$ is not an exact reconstruction of $\mathbf{x}$, rather it is probabilistic, which may generate $\mathbf{x}$ with a high probability. This results in an associated reconstruction error to be optimized as part of the training process :

$$L(\mathbf{x}, \mathbf{z}) \propto -\log p(\mathbf{x}|\mathbf{z}) \tag{11}$$

Typical choices for the loss function include the squared error objective for real-valued $\mathbf{x}$, or cross-entropy loss for binary $\mathbf{x}$. Note that it is possible to use any deterministic loss function, depending on the suitability with the target application. The autoencoder consists of estimating the parameters so as to minimize the reconstruction error. Intuitively, if a representation allows a good reconstruction of its input, it means that it has retained much of the information that was present in the input.

Traditional autoencoders have been known to perform almost as good as RBMs. The main advantage over RBMs is the simplicity of the training process as compared to the Energy-based model of the RBM. Moreover, they tend to have faster convergence. However, one critical issue is that in the absence of additional constraints, the autoencoder with a d-dimensional input and an encoding of dimension at least d could potentially just learn the identity function, for which many encodings would be useless i.e. just a copy of the input.

1) Denoising Autoencoders (DAE): In order to prevent traditional autoencoders from learning the identity function, implicit or explicit regularization of the weights may be applied, or additional sparsity constraints on the code may be introduced [18], [19]. RBMs have a very high capacity, and still not learn the identity function, since they try to capture the statistical structure in the input, by approximately maximizing the likelihood of a generative model.

Denoising autoencoders are a variant of traditional autoencoders that share this learning paradigm [42]. The denoising autoencoder minimizes the error in reconstructing the input from a stochastically corrupted version of the input. The corruption is achieved by adding random noise to the actual input. An alternative strategy, however, would be to add noise in the encoding process.

The intuition behind denoising autoencoders is based on the fact that a good representation is characterized by one that can be obtained robustly from a corrupted input and that will be useful for recovering the corresponding uncorrupted input. Performing the denoising task requires extracting features that capture useful structure in the input distribution. The denoising criterion enables us to use as many hidden units as necessary to capture the distribution, unlike in an Autoencoder.

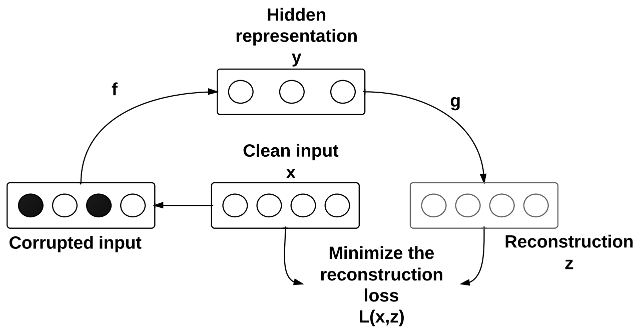

Fig. 4. Training algorithm for a Denoising Autoencoder

The Figure 4 depicts the training process for a DAE. The only difference with Autoencoders is that the input $\mathbf{x}$ is first corrupted to $\tilde{\mathbf{x}}$ before mapping to its hidden representation $\mathbf{y}$. Learning takes place using stochastic gradient descent. It can be shown that minimizing the stated reconstruction error (Equation 11) amounts to maximizing the mutual information between $\mathbf{x}$ and $\mathbf{y}$. There can be several types of corruptions and optimization criteria that can be taken into account depending upon the intrinsic nature of the problem [42].

The process of denoising can be interpreted as the *manifold assumption*, which states that natural high dimensional data concentrates close to a non-linear low-dimensional manifold. During the denoising training, we learn a stochastic operator $p(\mathbf{x}|\tilde{\mathbf{x}})$ that maps $\tilde{\mathbf{x}}$ back to its uncorrupted $\mathbf{x}$. Thus, a denoising autoencoder can be seen as a way to define and learn a manifold representing the data.

2) Stacked Denoising Autoencoders: The Stacked Denoising Autoencoder (SDAE) is an extension of the Stacked Autoencoder introduced in [5], [41]. The denoising autoencoders can also be stacked like RBMs to form a deep network by feeding the output representation of the denoising autoencoder found on the layer below as the input to the current layer. The *unsupervised pre-training* of such an architecture is done one layer at a time. Each layer is trained as an independent denoising autoencoder by minimizing the reconstruction error on the input from the previous layer.

Once all layers have been pre-trained, the network goes through a second stage known as *supervised fine-tuning* (similar to a DBN) where we minimize the prediction error on a supervised task. This is done by adding a logistic regression layer on top of the network, and then training the entire network as a single multilayer perceptron. The SDAE algorithm for training deep networks is known to yield good classification performance on many standard benchmark problems due to the representations extracted layer by layer, using a purely unsupervised local denoising criterion. Moreover, it also learns feature extractors representing useful structure in the data that regular autoencoders are unable to learn [42].

E. Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are variants of Multi Layer Perceptrons which draw inspiration from biology. The mammalian brain is known to have a complex arrangement of cells in the visual cortex [20]. The visual cortex being the most powerful "vision" system in existence, it is logical to emulate its behavior [14], [27], [38]. The cells are sensitive to small sub-regions of the input space i.e. receptive fields. These fields are tied in such a way so as to cover the entire visual field. These filters are local in the input space and are thus better suited to exploit the strong spatial correlation properties present in natural images.

CNNs are typically deep i.e. have more than 3 layers and exploit spatial correlation by enforcing local connectivity patterns between neurons of adjacent layers. In contrast to Multi-Layer Perceptrons, which have all-to-all connectivity, the input hidden neurons in the m^{th} layer are connected only to a local subset of the neurons in the $(m+1)^{th}$ layer. An example structure is shown in the Figure 5.

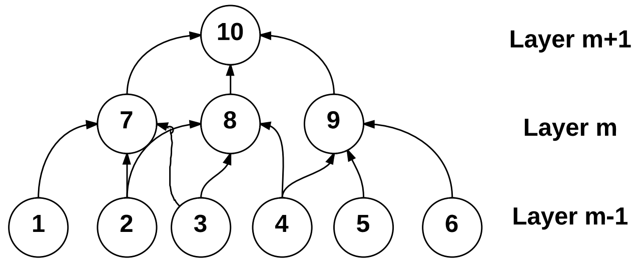

Fig. 5. The connectivity between neurons in adjacent layers in CNNs [2].

The neurons in layer m in the Figure 5 have a receptive field of 3, and are hence connected to 3 neurons from the layer $m-1$. The architecture thus confines the learnt filters to comprise of spatially local patterns.

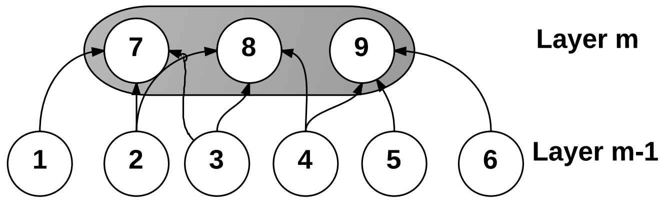

Fig. 6. Shared weights across the entire visual field in CNNs [2].

Each sparse filter h_i is additionally replicated across the entire visual field. These replicated units form a feature map, and share the same weight vector and bias (shown by the shaded region in Figure 6). Replicating units in this manner allows for features to be detected regardless of their position in the visual field. Moreover, it greatly reduces the number of free parameters to be learnt. Stacking many such layers one above the other leads to non-linear filters which ultimately encompass the entire vision field. As the input moves higher up in the hierarchy, it gets transformed into more invariant representations. Thus, a fast changing pattern at the bottom of this hierarchy becomes a nearly invariant pattern at the top. As a result, CNNs achieve better generalization on vision problems.

F. Sparse Distributed Representations & Sparse Coding

Sparse Distributed Representation (SDR) is a mathematical model of the human long-term memory [22]. It is primarily used for storing and retrieving large amounts of information without focusing on the accuracy of the information. It uses patterns to serve as memory addresses, where information is retrieved based on similarities between addresses. A typical characteristic of such representations is that they are very sparse i.e. they have a large number of bits, but only a few are active at a time. Each bit represents some attribute of the class of objects or concepts being encoded.

Such representations are much longer (e.g. 2000 bits) as compared to typical ASCII (8 bits) or Unicode (16 bits) encodings. However, a major difference between the former and the latter is the meaning associated with every bit in the code. In case of ASCII or Unicode, an individual bit has no meaning and one has to look at the entire code to infer a meaning. However, each bit in a SDR has a semantic meaning associated with it which can be inferred even by a single bit activation. Such representations have certain key properties which make them a good knowledge representation scheme [15] :

- It is possible to directly compare two SDRs with bit comparisons. The shared bits directly map to the semantic similarity between the corresponding objects.
- SDRs can be stored very efficiently by storing only the indices of the active bits, rather than storing the entire representations. Moreover, subsampling i.e. storing only a part of the active bits results only in minimal loss in the richness of the representation.
- It is very efficient to detect membership of a given SDR in a group of SDRs by simply comparing the SDR bits with those of the group's union. Although such a technique is probabilistic in nature, it is possible to show mathematically that given a sufficiently large SDR, matching done using such a technique is almost guaranteed to be correct.

1) Sparse Coding: Sparse coding is a type of neural code which is based on sparse distributed representations. The goal of sparse coding is to represent input vectors as a sparse approximate weighted linear combination of basis vectors [43]. For an input vector $\mathbf{y} \in \mathbb{R}^k$,

$$\mathbf{y} \approx \sum_j b_j s_j = Bs \tag{12}$$

where $b_1,\ldots,b_n \in \mathbb{R}^k$ and $s \in \mathbb{R}^n$ is a sparse vector of coefficients. Unlike similar methods such as PCA, the basis set B founded in sparse coding can be overcomplete ($n > k$), and can represent nonlinear features of the training set $\mathbf{x}$. Let $X \in R^{k\times m}$ be the input matrix (each column is an input vector), let $B \in R^{k\times n}$ be the basis matrix (each column is a basis vector), and let $S \in R^{n\times m}$ be the coefficient matrix (each column is a coefficient vector). Then the optimization problem can be written as (cite efficient sparse coding techniques) :

$$\min_{B,S} \quad ||X - BS||_2^2 + \beta \sum_{i,j} ||S_{i,j}||_1 \tag{13}$$

$$\text{s.t} \quad \sum_i B_{i,j}^2 \le c, \qquad \forall j = 1,\ldots,n \tag{14}$$

Algorithm 1: Self-taught Learning via Sparse Coding [45]

```
Input  : Labeled training set T = $(x_l^{(1)}, y_l^{(1)}),\ldots,(x_l^{(m)}, y_l^{(m)})$.
Input  : Unlabeled data $x_u^{(1)},\ldots,x_u^{(k)}$.
1 Using unlabeled data $x_u^{(i)}$, solve the optimization problem
  13 to obtain the bases $B$.
2 Compute features for the classification task to obtain a
  new labeled training set $\hat{T}$ using the bases $B$ and the
  sparse vectors $s^{(i)}$.
3 Learn a classifier $C$ by applying a supervised learning
  algorithm to the labeled training set $\hat{T}$.
  Output: Learned classifier for the classification task.
```

Semi-supervised learning using sparse coding could be extended to multiple layers to train a hierarchy of features using a supervised classifier at the top [23].

G. Hierarchical Temporal Memory

Hierarchical Temporal Memory (HTM) is another technology modeled the mammalian neocortex [15]. The neocortex is the seat of intelligent thought in the mammalian brain. Given the diverse suite of cognitive functions, one might expect the neocortex to implement an equally diverse suite of specialized neural algorithm. However, research indicates that the neocortex displays a remarkably uniform pattern of neural circuitry [7], [44]. The biological evidence suggests that the neocortex implements a common set of algorithms to perform many different intelligent functions.

HTM provides a theoretical framework for understanding the neocortex and its many capabilities. HTM models neurons, which are arranged in columns, layers, regions, and in a hierarchy. HTMs can therefore be regarded as a new form of neural networks. Programming HTMs is unlike programming traditional computers. HTMs are trained through exposure to a stream of sensory data, unlike today's computers [15]. Its capabilities are determined by what it has been exposed to.

HTM is fundamentally a memory based system that learns in space as well as time. HTM networks are trained on lots of time varying data, and rely on storing a large set of patterns and sequences. The manner in which data is stored is logically different from the standard model used in computers today. Classic computer memory has a flat organization and does not have an inherent notion of time. HTM memory has a hierarchical organization and is inherently tie based. Information is always stored in a distributed fashion. One can control the size of the hierarchy and what to train the system on, but the HTM controls where and how the information is stored.

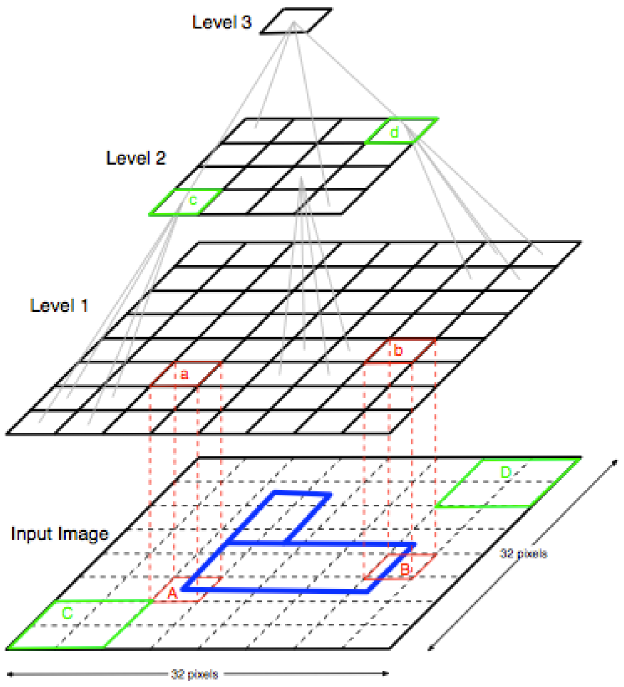

Fig. 7. An example of a HTM hierarchy for image recognition [12]. The training is done using videos of the training images. These consist of vertical and horizontal translations, scale and rotate transformations.

The Figure 7 shows a HTM which comprises tree-shaped hierarchy of levels that are composed of elements called nodes or cells. Each HTM node has the same basic functionality. A single level in the hierarchy is known as a region. Higher hierarchy levels often have fewer nodes and therefore encompass a greater portion of the field of view. Higher levels can also reuse patterns learnt at the lower levels and memorize sequences of more complex patterns. Each HTM region learns by identifying and memorizing spatial patterns and later on identifying temporal sequences of spatial patterns that are likely to occur one after another. Level 1 and Level 2 nodes are trained using a time varying input that is representative of the entire input domain. For e.g., for an image classification problem, the training pattern might be videos of the training images, consisting of translate, rotate and scale transformations. The node at the top level is then trained using supervision i.e. a label is associated with every training image presented in every position.

The Figure 8 shows the structure of a single node and the inference process in a HTM network, applied to a computer vision problem. During training, the node receives a temporal

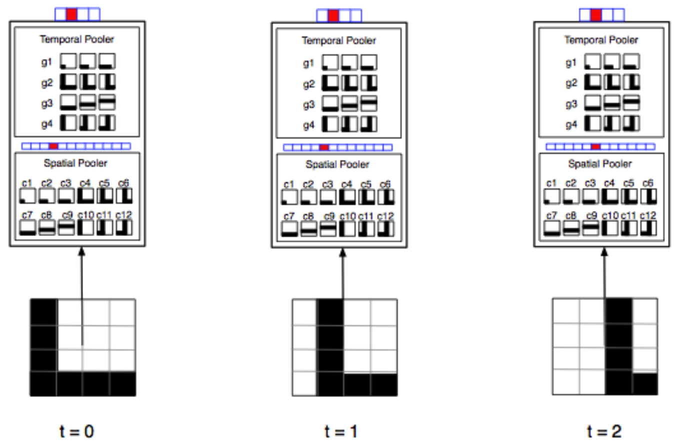

Fig. 8. A fully-learned node in inference mode for 3 time steps [12]. The node has learnt 12 quantization centers within its spatial pooler and 4 temporal groups or sequences within its temporal pooler.

sequence of spatial patterns as input. The node learning algorithm consists of two stages [15] :

1) *Spatial Pooling:* This identifies frequently observed patterns and stores them as quantization centers. Patterns that are very similar to each other are treated as one. Therefore, a large number of possible input patterns are reduced to a manageable number of known quantization centers.

2) *Temporal Pooling:* This partitions quantization centers into temporal groups based on their likeliness to follow each other in the training sequences using a greedy method [15]. Each group represents a *cause* of the input pattern [14].

Once all the nodes on one layer have been trained, they are operated in inference mode to train the nodes in the next higher layer. This is shown with a 'L' pattern moving rightwards over three timesteps in the Figure 8. At each timestep, the spatial pooler produces a SDR (See Section IV-F) with one of its learnt quantization centers active. This is passed on to the temporal pooler which associates a group or sequence with this pattern. As is evident from the Figure 8, this moving pattern is recognized as part of a single sequence in the output of the temporal pooler. Depending upon the connections, the outputs of these nodes are concatenated to form input patterns for the nodes in the next higher layer.

Upon convergence, the entire HTM network is operated in inference mode. Here, each node calculates the probability that a given pattern belongs to a known temporal sequence. As this information propagates upwards in the hierarchy, it forms higher levels of belief about the pattern. This process is done in an entirely unsupervised manner. Depending upon the problem, a classifier can be attached to the top level node in the network to associate appropriate labels with the input patterns.

V. Applications

This section describes selected interesting applications of deep learning methods to supervised and unsupervised learning problems reported in literature. The goal here is not to perform an exhaustive comparison between deep learning methods, but rather to demonstrate that these methods achieve better results over a wide range of benchmarks with different data characteristics.

The first real-world experiments were done in the field of Computer Vision using SVMs [31] and exhibited better performance as compared to the then state-of-the-art results achieved with multilayer perceptrons. However, the recent class of deep learning algorithms [5], [19], [42] have reversed the situation.

The Table I shows the test errors obtained on 10 different classificiation problems [42] comparing the performance of SVMs with single-layered and multi-layered DBNs, SAEs and SDAEs. The datasets consist of the standard MNIST digit classification problem [25] with 60000 training examples, its challenging variations (with rotations, random noise and image backgrounds) as well as a variation of the *tzanetakis* audio genre classification dataset [6] which contains 10000 three-second audio clips, equally distributed among 10 musical genres.

Dataset	**SVM**$_{rbf}$	**DBN-1**	**DBN-3**	**SDAE-3**
MNIST	1.40±0.23	1.21±0.21	**1.24**±0.22	1.28±0.22
basic	3.03±0.15	3.94±0.17	3.11±0.15	**2.84**±0.15
rot	11.11±0.28	14.69±0.31	10.30±0.27	**9.53**±0.26
bg-rand	14.58±0.31	9.80±0.26	**6.73**±0.22	10.30±0.27
bg-img	22.61±0.37	16.15±0.32	**16.31**±0.32	16.68±0.33
bg-img-rot	55.18±0.44	52.21±0.44	47.39±0.44	**43.76**±0.43
rect	2.15±0.13	4.71±0.19	2.60±0.14	**1.99**±0.12
rect-img	24.04±0.37	23.69±0.37	22.50±0.37	**21.59**±0.36
convex	19.13±0.34	19.92±0.35	18.63±0.34	**19.06**±0.34
tzanetakis	**14.41**±2.18	18.07±1.31	18.38±1.64	16.02±1.04

TABLE I
Comparison of SDAE and DBN with SVM [42]. Best performer is shown in bold. DBN-1 is a single layered DBN. DBN-3 and SDAE-3 are three layered structures. SDAE and DBN outperform SVM for all datasets except the *tzanetakis* audio dataset.

It can be observed that SDAE-3 systematically outperforms the baseline SVM. For all but one problem, SDAE-3 is either the best performing algorithm or has its confidence interval overlap with that of the winning algorithm. The difference in performance between DBN-1 and SDAE-3 (or DBN-3) is indicative of the capability of deep networks in extracting high-level information from the input data. In most cases, 3 layers of denoising autoencoder is on par or better than stacking 3 layers of RBMs in DBN-3. Numerous comparative studies [26], [32], [34] have also been conducted on variants of the MNIST dataset using HTMs, CNNs and Sparse Coding methods. These methods are generally known to perform better or as good as SVMs. HTMs are especially better at time-based inference where the input is a dynamic pattern.

DBNs have also been applied to Information Retrieval problems. Semantic hashing [37] is a technique that produces a shortlist of similar documents in a time that is independent of the size of the document collection and linear in the size of the shortlist. Moreover, only a few machine instructions are required per document in the shortlist. Every document is encoded as a frequency vector of the words present in the document. The document set is used for learning a stack of RBMs in which the feature activations of one RBM are treated as data by the next RBM. This process semantically maps a 2000 feature frequency vector into a 32-bit binary code. After pre-training the RBMs are "unrolled" to create a multi-layer autoencoder that is fine-tuned using backpropagation. Documents similar to a query document are then found by simply accessing all the addresses that differ by only a few bits from the address of the query document. This method achieves higher accuracy than applying TF-IDF (Term Frequency-Inverse Document Frequency) to the entire document set and is about 20-50x faster than Locality-Sensitive Hashing [9], which is the fastest current method.

Automatic Speech Recognition (ASR) is another important application area. State-of-the-art ASR systems typically use Hidden Markov Models (HMMs) to model the sequential structure of speech signals, with local spectral varaibliity modeled using mixtures of Gaussian densities. DBNs have recently proved to be very effective for such kind of acoustic modeling. On the standard TIMIT corpus, DBNs have been shown to consistently outperform other techniques and the best DBNs achieve a Phone Error Rate (PER) of 23.0% on the TIMIT core test set [33].

Natural Language Processing (NLP) techniques are used for information extraction, machine translation, summarization, search and human-computer interfaces. While complete semantic understanding is still a far-distant goal, CNNs have been used to define a unified architecture for NLP that learns features that are relevant to the tasks at hand given very limited prior knowledge [8]. This is in contrast to the divide and conquer approach of identifying several sub-tasks useful for application development and analysis, such as part-of-speech tagging, chunking and parsing, word-sense disambiguation, semantic-role labeling and others. This is achieved by training the entire network jointly on all these tasks using weight-sharing, a technique known as *multitask learning*. All these tasks use labeled data except the language model which is learnt from unlabeled text and represents a novel form of *semi-supervised learning* for shared tasks. The synergy of these two techniques improves the generalization of the shared tasks, and results in state-of-the-art performance.

VI. Challenges and Future Work

There has been a great deal of recent work on learning useful representations of data with deep learning algorithms. These algorithms have shown promise as a means of learning invariant representations of data. Despite the recent successes, many existing hierarchical models are still far from being able to represent, identify and learn the wide variety of possible patterns and structure in real-world data. Existing models cannot cope with new tasks for which they have not been specifically trained. Massive volumes of training data, high-dimensional input spaces and multiple categories (in the order of several tens of thousands) pose challenging questions on how to effectively train deep models.

Moreover, learning of deep architectures is still an open problem in itself. There is currently a limited amount of understanding about why deep learning algorithms perform so well on certain tasks. Most such algorithms applied to supervised learning problems often involve an unsupervised pre-training phase. Although there have been severa divulging explanations on why unsupervised pre-training helps [5], [10], [11], this is still an open question and an area of active research. There is still a significant amount of work to be done in tuning deep model configurations (for e.g. number of layers, size of each layer etc.) to best suit the target problems, something which is currently done in an ad-hoc fashion.

Another significant challenge is to develop systems that can detect multiple high-order co-occurrences. i.e. situations where two different entities (words, documents, values) are contextually relevant. Most deep learning algorithms have biological motivations, specifically known as the Cortical Learning Algorithms (CLA). Characterization of the capacity of these algorithms, implementing them in hierarchies, and understanding how to integrate multiple input data streams (corresponding to integration of multiple sensory inputs in the mammalian brain) are open problems in the field of deep learning.

Cognitive Computing is a new and exciting area of research [30] that has come about as a result of advances in deep learning. It deals with making machines think like humans. Using advanced algorithms and silicon circuitry, the aim is to develop cognitive systems that can learn through experiences, find correlations, create hypotheses, and remember - and learn from - the outcomes. Commercial systems like Grok [21] are also being developed, based on deep learning principles, with the goal of providing organizations with analytical expertise to handle the explosive growth of data.

VII. Conclusion

The goal of the present report was to give an introduction into the field of deep learning. We presented a brief overview of the deep learning approach and how it differs from the traditional approach to AI. We highlighted the major motivations behind the concept of deep learning. We looked into the historical development of learning architectures, outlined their drawbacks and stated recent breakthroughs in learning deep architectures. We discussed the major deep learning algorithms along with some recent and interesting applications of these algorithms to supervised and unsupervised learning problems such as Computer Vision, Information Retrieval, Speech Recognition and Natural Language Processing. Finally, we presented some of the underlying challenges in this field, current trends and the scope for future research work. In conclusion, deep learning is a very promsing field of AI that is very much in its infancy. Research in the past decade has provided a good foundation to build upon and move closer towards the ultimate goal of building truly intelligent machines.

References

[1] David H. Ackley, Geoffrey E. Hinton, and Terrence J. Sejnowski. A learning algorithm for boltzmann machines. *Cognitive Science*, 9(1):147–169, 1985.

[2] Yoshua Bengio. Deep learning tutorials, 2008.

[3] Yoshua Bengio. Learning deep architectures for AI. *Foundations and Trends in Machine Learning*, 2(1):1–127, 2009. Also published as a book. Now Publishers, 2009.

[4] Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle. Greedy layer-wise training of deep networks. *Advances in Neural Information Processing Systems*, 19:153, 2007.

[5] Yoshua Bengio and Yann LeCun. Scaling learning algorithms towards ai. *Large-Scale Kernel Machines*, 34, 2007.

[6] James Bergstra. *Algorithms for classifying recorded music by genre*. PhD thesis, Citeseer, 2006.

[7] Matthew M Botvinick, Yael Niv, and Andrew C Barto. Hierarchically organized behavior and its neural foundations: A reinforcement learning perspective. *Cognition*, 113(3):262–280, 2009.

[8] Ronan Collobert and Jason Weston. A unified architecture for natural language processing: deep neural networks with multitask learning. In *Proceedings of the 25th International Conference on Machine Learning*, pages 160–167. ACM, 2008.

[9] Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the 20th Annual Symposium on Computational Geometry*, pages 253–262. ACM, 2004.

[10] Dumitru Erhan, Yoshua Bengio, Aaron Courville, Pierre-Antoine Manzagol, Pascal Vincent, and Samy Bengio. Why does unsupervised pre-training help deep learning? *Journal of Machine Learning Research*, 11:625–660, March 2010.

[11] Dumitru Erhan, Pierre-Antoine Manzagol, Yoshua Bengio, Samy Bengio, and Pascal Vincent. The difficulty of training deep architectures and the effect of unsupervised pre-training. In *Proceedings of The 12th International Conference on Artificial Intelligence and Statistics (AISTATS09)*, pages 153–160. Citeseer, 2009.

[12] Dileep George and Bobby Jaros. The htm learning algorithms. Technical report, Numenta Inc., 2007.

[13] Patricia M Greenfield et al. Language, tools and brain: The ontogeny and phylogeny of hierarchically organized sequential behavior. *Behavioral and Brain Sciences*, 14(4):531–551, 1991.

[14] Jeff Hawkins and Sandra Blakslee. *On intelligence*. Times Books, New York, 2004.

[15] Jeff Hawkins and Dileep George. The HTM Cortical Learning Algorithms, September 2011.

[16] GE Hinton, P Dayan, BJ Frey, and RM Neal. The "wake-sleep" algorithm for unsupervised neural networks. *Science*, 268(5214):1158–1161, 1995.

[17] Geoffrey Hinton. A practical guide to training restricted boltzmann machines. *Momentum*, 9:1, 2010.

[18] Geoffrey E Hinton. Learning multiple layers of representation. *Trends in Cognitive Sciences*, 11(10):428–434, 2007.

[19] Geoffrey E. Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18(7):1527–1554, July 2006.

[20] David H Hubel and Torsten N Wiesel. Receptive fields and functional architecture of monkey striate cortex. *The Journal of Physiology*, 195(1):215–243, 1968.

[21] Numenta Inc. Grok : Action Intelligence for Fast Data. https://www.numenta.com/product.html/, 2013.

[22] Pentti Kanerva. *Sparse distributed memory*. MIT Press, 1988.

[23] Koray Kavukcuoglu, Marc'Aurelio Ranzato, and Yann LeCun. Fast inference in sparse coding algorithms with applications to object recognition. *CoRR*, abs/1010.3467, 2010.

[24] Fabien Lauer and Gérard Bloch. Incorporating prior knowledge in support vector machines for classification: A review. *Neurocomputing*, 71(7):1578–1594, 2008.

[25] Y LeCun. Mnist dataset, 2000.

[26] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. In *Intelligent Signal Processing*, pages 306–351. IEEE Press, 2001.

[27] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[28] Christopher Poultney MarcAurelio Ranzato, Sumit Chopra, and Yann LeCun. Efficient learning of sparse representations with an energy-based model. *Advances in Neural Information Processing Systems*, 19:1137–1144, 2006.

[29] Marvin Minsky and Papert Seymour. *Perceptrons.* MIT press, 1969.

[30] Dharmendra S. Modha, Rajagopal Ananthanarayanan, Steven K. Esser, Anthony Ndirango, Anthony J. Sherbondy, and Raghavendra Singh. Cognitive computing. *Communications of the ACM*, 54(8):62–71, August 2011.

[31] Klaus-Robert Muller, Sebastian Mika, Gunnar Ratsch, Koji Tsuda, and Bernhard Scholkopf. An introduction to kernel-based learning algorithms. *Neural Networks, IEEE Transactions on*, 12(2):181–201, 2001.

[32] Ryan William Price. *Hierarchical Temporal Memory Cortical Learning Algorithm for Pattern Recognition on Multi-core Architectures*. PhD thesis, Portland State University, 2011.

[33] Abdel rahman Mohamed, George E. Dahl, and Geoffrey E. Hinton. Deep belief networks for phone recognition. In *NIPS Workshop on Deep Learning for Speech Recognition and Related Applications*, 2009.

[34] Marc'Aurelio Ranzato, Y-Lan Boureau, and Yann LeCun. Sparse feature learning for deep belief networks. In *Advances in Neural Information Processing Systems (NIPS 2007)*, volume 20, 2007.

[35] F Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408iii, 1958.

[36] DE Rummelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *Nature*, 323(9):533–535, 1986.

[37] Ruslan Salakhutdinov and Geoffrey Hinton. Semantic hashing. *International Journal of Approximate Reasoning*, 50(7):969–978, 2009.

[38] Thomas Serre, Lior Wolf, Stanley Bileschi, Maximilian Riesenhuber, and Tomaso Poggio. Robust object recognition with cortex-like mechanisms. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 29(3):411–426, 2007.

[39] David S. Touretzky and Geoffrey E. Hinton. Symbols among the neurons: Details of a connectionist inference architecture. In *IJCAI*, pages 238–243, 1985.

[40] Alan Mathison Turing. On computable numbers, with an application to the entscheidungsproblem. a correction. *Proceedings of the London Mathematical Society*, 2(1):544, 1938.

[41] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th International Conference on Machine learning*, pages 1096–1103. ACM, 2008.

[42] Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *The Journal of Machine Learning Research*, 11:3371–3408, 2010.

[43] Shuicheng Yan and Huan Wang. Semi-supervised learning by sparse representation. In *SIAM International Conference on Data Mining, SDM*, pages 792–801, 2009.

[44] Changsong Zhou, Lucia Zemanová, Gorka Zamora, Claus C Hilgetag, and Jürgen Kurths. Hierarchical organization unveiled by functional connectivity in complex brain networks. *Physical Review Letters*, 97(23):238103, 2006.

[45] David Ziegler. Semi-supervised learning with sparse distributed representations, 2009.