On the Origin of Deep Learning

Haohan Wang haohanw@cs.cmu.edu

Bhiksha Raj bhiksha@cs.cmu.edu

Language Technologies Institute
School of Computer Science
Carnegie Mellon University

Eric P. Xing epxing@cs.cmu.edu

Machine Learning Department
School of Computer Science
Carnegie Mellon University

Abstract

This paper is a review of the evolutionary history of deep learning models. It covers from the genesis of neural networks when associationism modeling of the brain is studied, to the models that dominate the last decade of research in deep learning like convolutional neural networks, deep belief networks, and recurrent neural networks, and extends to popular recent models like variational autoencoder and generative adversarial nets. In addition to a review of these models, this paper primarily focuses on the precedents of the models above, examining how the initial ideas are assembled to construct the early models and how these preliminary models are developed into their current forms. Many of these evolutionary paths last more than half a century and have a diversity of directions. For example, CNN is built on prior knowledge of biological vision system; DBN is evolved from a trade-off of modeling power and computation complexity of graphical models and many nowadays models are neural counterparts of ancient linear models. This paper reviews these evolutionary paths and offers a concise thought flow of how these models are developed, and aims to provide a thorough background for deep learning. More importantly, along with the path, this paper summarizes the gist behind these milestones and proposes many directions to guide the future research of deep learning.

1. Introduction

Deep learning has dramatically improved the state-of-the-art in many different artificial intelligent tasks like object detection, speech recognition, machine translation (LeCun et al., 2015). Its deep architecture nature grants deep learning the possibility of solving many more complicated AI tasks (Bengio, 2009). As a result, researchers are extending deep learning to a variety of different modern domains and tasks in additional to traditional tasks like object detection, face recognition, or language models, for example, Osako et al. (2015) uses the recurrent neural network to denoise speech signals, Gupta et al. (2015) uses stacked autoencoders to discover clustering patterns of gene expressions. Gatys et al. (2015) uses a neural model to generate images with different styles. Wang et al. (2016) uses deep learning to allow sentiment analysis from multiple modalities simultaneously, etc. This period is the era to witness the blooming of deep learning research.

However, to fundamentally push the deep learning research frontier forward, one needs to thoroughly understand what has been attempted in the history and why current models exist in present forms. This paper summarizes the evolutionary history of several different deep learning models and explains the main ideas behind these models and their relationship to the ancestors. To understand the past work is not trivial as deep learning has evolved over a long time of history, as showed in Table 1. Therefore, this paper aims to offer the readers a walk-through of the major milestones of deep learning research. We will cover the milestones as showed in Table 1, as well as many additional works. We will split the story into different sections for the clearness of presentation.

This paper starts the discussion from research on the human brain modeling. Although the success of deep learning nowadays is not necessarily due to its resemblance of the human brain (more due to its deep architecture), the ambition to build a system that simulate brain indeed thrust the initial development of neural networks. Therefore, the next section begins with connectionism and naturally leads to the age when shallow neural network matures.

With the maturity of neural networks, this paper continues to briefly discuss the necessity of extending shallow neural networks into deeper ones, as well as the promises deep neural networks make and the challenges deep architecture introduces.

With the establishment of the deep neural network, this paper diverges into three different popular deep learning topics. Specifically, in Section 4, this paper elaborates how Deep Belief Nets and its construction component Restricted Boltzmann Machine evolve as a trade-off of modeling power and computation loads. In Section 5, this paper focuses on the development history of Convolutional Neural Network, featured with the prominent steps along the ladder of ImageNet competition. In Section 6, this paper discusses the development of Recurrent Neural Networks, its successors like LSTM, attention models and the successes they achieved.

After an extensive discussion of the traditional deep learning family, this paper proceeds to more recent topics like Variational AutoEncoders (VAE) and Generative Adversarial Networks (GAN). The modern models do not typically have a long evolutionary history in neural networks domain but are inherited from ancestors from other machine learning topics like factor analysis, probabilistic graphical models etc. This paper also explains the connection of these modern models to their ancestors and conveys the idea that while we

Table 1: Major milestones that will be covered in this paper

Year	Contributer	Contribution
300 BC	Aristotle	introduced Associationism, started the history of human's attempt to understand brain.
1873	Alexander Bain	introduced Neural Groupings as the earliest models of neural network, inspired Hebbian Learning Rule.
1943	McCulloch & Pitts	introduced MCP Model, which is considered as the ancestor of Artificial Neural Model.
1949	Donald Hebb	considered as the father of neural networks, introduced Hebbian Learning Rule, which lays the foundation of modern neural network.
1958	Frank Rosenblatt	introduced the first perceptron, which highly resembles modern perceptron.
1974	Paul Werbos	introduced Backpropagation
1980	Teuvo Kohonen	introduced Self Organizing Map
	Kunihiko Fukushima	introduced Neocogitron, which inspired Convolutional Neural Network
1982	John Hopfield	introduced Hopfield Network
1985	Hilton & Sejnowski	introduced Boltzmann Machine
1986	Paul Smolensky	introduced Harmonium, which is later known as Restricted Boltzmann Machine
	Michael I. Jordan	defined and introduced Recurrent Neural Network
1990	Yann LeCun	introduced LeNet, showed the possibility of deep neural networks in practice
1997	Schuster & Paliwal	introduced Bidirectional Recurrent Neural Network
	Hochreiter & Schmidhuber	introduced LSTM, solved the problem of vanishing gradient in recurrent neural networks
2006	Geoffrey Hinton	introduced Deep Belief Networks, also introduced layer-wise pretraining technique, opened current deep learning era.
2009	Salakhutdinov & Hinton	introduced Deep Boltzmann Machines
2012	Geoffrey Hinton	introduced Dropout, an efficient way of training neural networks
2013	Kingma & Welling	introduced Variational Autoencoder (VAE), which may bridge the field of deep learning and the field of Bayesian probabilistic graphic models.
2014	Ian J. Goodfellow	introduced Generative Adversarial Network.
2015	Ioffe & Szegedy	introduced Batch Normalization

admit the significant improvement these models contribute to deep learning society, similar ideas have been seen in the past.

While this paper primarily discusses deep learning models, optimization of deep architecture is an inevitable topic in this society. Section 8 is devoted to a brief summary of optimization techniques, including advanced gradient method, Dropout, Batch Normalization, etc.

This paper could be read as a complementary of (Schmidhuber, 2015). Schmidhuber's paper is aimed to assign credit to all those who contributed to the present state of the art, so his paper focuses on every single incremental work along the path, therefore cannot elaborate well enough on each of them. On the other hand, our paper is aimed at providing the background for readers to understand how these models are developed. Therefore, we emphasize on the milestones and elaborate those ideas to help build associations between these ideas. In addition to the paths of classical deep learning models in (Schmidhuber, 2015), we also discuss those recent deep learning work that builds from classical linear models. Another article that readers could read as a complementary is (Anderson and Rosenfeld, 2000) where the authors conducted extensive interviews with well-known scientific leaders in the 90s on the topic of the neural networks' history.

2. From Aristotle to Modern Artificial Neural Networks

The study of deep learning and artificial neural networks originates from our ambition to build a computer system simulating the human brain. To build such a system requires understandings of the functionality of our cognitive system. Therefore, this paper traces all the way back to the origins of attempts to understand the brain and starts the discussion of Aristotle's Associationism around 300 B.C.

2.1 Associationism

> "When, therefore, we accomplish an act of reminiscence, we pass through a certain series of precursive movements, until we arrive at a movement on which the one we are in quest of is habitually consequent. Hence, too, it is that we hunt through the mental train, excogitating from the present or some other, and from similar or contrary or coadjacent. Through this process reminiscence takes place. For the movements are, in these cases, sometimes at the same time, sometimes parts of the same whole, so that the subsequent movement is already more than half accomplished."

This remarkable paragraph of Aristotle is seen as the starting point of Associationism (Burnham, 1888). Associationism is a theory states that mind is a set of conceptual elements that are organized as associations between these elements. Inspired by Plato, Aristotle examined the processes of remembrance and recall and brought up with four laws of association (Boeree, 2000).

- Contiguity: Things or events with spatial or temporal proximity tend to be associated in the mind.
- Frequency: The number of occurrences of two events is proportional to the strength of association between these two events.
- Similarity: Thought of one event tends to trigger the thought of a similar event.
- Contrast: Thought of one event tends to trigger the thought of an opposite event.

Back then, Aristotle described the implementation of these laws in our mind as common sense. For example, the feel, the smell, or the taste of an apple should naturally lead to the concept of an apple, as common sense. Nowadays, it is surprising to see that these laws proposed more than 2000 years ago still serve as the fundamental assumptions of machine learning methods. For example, samples that are near each other (under a defined distance) are clustered into one group; explanatory variables that frequently occur with response variables draw more attention from the model; similar/dissimilar data are usually represented with more similar/dissimilar embeddings in latent space.

Contemporaneously, similar laws were also proposed by Zeno of Citium, Epicurus and St Augustine of Hippo. The theory of associationism was later strengthened with a variety of philosophers or psychologists. Thomas Hobbes (1588-1679) stated that the complex experiences were the association of simple experiences, which were associations of sensations. He also believed that association exists by means of coherence and frequency as its strength

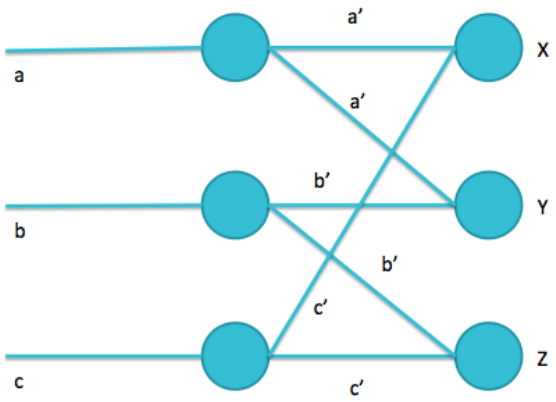


Figure 1: Illustration of neural groupings in (Bain, 1873)

factor. Meanwhile, John Locke (1632-1704) introduced the concept of "association of ideas". He still separated the concept of ideas of sensation and ideas of reflection and he stated that complex ideas could be derived from a combination of these two simple ideas. David Hume (1711-1776) later reduced Aristotle's four laws into three: resemblance (similarity), contiguity, and cause and effect. He believed that whatever coherence the world seemed to have was a matter of these three laws. Dugald Stewart (1753-1828) extended these three laws with several other principles, among an obvious one: accidental coincidence in the sounds of words. Thomas Reid (1710-1796) believed that no original quality of mind was required to explain the spontaneous recurrence of thinking, rather than habits. James Mill (1773-1836) emphasized on the law of frequency as the key to learning, which is very similar to later stages of research.

David Hartley (1705-1757), as a physician, was remarkably regarded as the one that made associationism popular (Hartley, 2013). In addition to existing laws, he proposed his argument that memory could be conceived as smaller scale vibrations in the same regions of the brain as the original sensory experience. These vibrations can link up to represent complex ideas and therefore act as a material basis for the stream of consciousness. This idea potentially inspired Hebbian learning rule, which will be discussed later in this paper to lay the foundation of neural networks.

2.2 Bain and Neural Groupings

Besides David Hartley, Alexander Bain (1818-1903) also contributed to the fundamental ideas of Hebbian Learning Rule (Wilkes and Wade, 1997). In this book, Bain (1873) related the processes of associative memory to the distribution of activity of *neural groupings* (a term that he used to denote neural networks back then). He proposed a constructive mode of storage capable of assembling what was required, in contrast to alternative traditional mode of storage with prestored memories.

To further illustrate his ideas, Bain first described the computational flexibility that allows a neural grouping to function when multiple associations are to be stored. With a few hypothesis, Bain managed to describe a structure that highly resembled the neural

networks of today: an individual cell is summarizing the stimulation from other selected linked cells within a grouping, as showed in Figure 1. The joint stimulation from a and c triggers X, stimulation from b and c triggers Y and stimulation from a and c triggers Z. In his original illustration, a, b, c stand for simulations, X and Y are outcomes of cells.

With the establishment of how this associative structure of neural grouping can function as memory, Bain proceeded to describe the construction of these structures. He followed the directions of associationism and stated that relevant impressions of neural groupings must be made in temporal contiguity for a period, either on one occasion or repeated occasions.

Further, Bain described the computational properties of neural grouping: connections are strengthened or weakened through experience via changes of intervening cell-substance. Therefore, the induction of these circuits would be selected comparatively strong or weak.

As we will see in the following section, Hebb's postulate highly resembles Bain's description, although nowadays we usually label this postulate as Hebb's, rather than Bain's, according to (Wilkes and Wade, 1997). This omission of Bain's contribution may also be due to Bain's lack of confidence in his own theory: Eventually, Bain was not convinced by himself and doubted about the practical values of neural groupings.

2.3 Hebbian Learning Rule

Hebbian Learning Rule is named after Donald O. Hebb (1904-1985) since it was introduced in his work *The Organization of Behavior* (Hebb, 1949). Hebb is also seen as the father of Neural Networks because of this work (Didier and Bigand, 2011).

In 1949, Hebb stated the famous rule: "Cells that fire together, wire together", which emphasized on the activation behavior of co-fired cells. More specifically, in his book, he stated that:

> "When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that As efficiency, as one of the cells firing B, is increased."

This archaic paragraph can be re-written into modern machine learning languages as the following:

$$\Delta w_i = \eta x_i y \tag{1}$$

where Δw_i stands for the change of synaptic weights (w_i) of Neuron i, of which the input signal is x_i. y denotes the postsynaptic response and η denotes learning rate. In other words, Hebbian Learning Rule states that the connection between two units should be strengthened as the frequency of co-occurrences of these two units increase.

Although Hebbian Learning Rule is seen as laying the foundation of neural networks, seen today, its drawbacks are obvious: as co-occurrences appear more, the weights of connections keep increasing and the weights of a dominant signal will increase exponentially. This is known as the unstableness of Hebbian Learning Rule (Principe et al., 1999). Fortunately, these problems did not influence Hebb's identity as the father of neural networks.

2.4 Oja's Rule and Principal Component Analyzer

Erkki Oja extended Hebbian Learning Rule to avoid the unstableness property and he also showed that a neuron, following this updating rule, is approximating the behavior of a Principal Component Analyzer (PCA) (Oja, 1982).

Long story short, Oja introduced a normalization term to rescue Hebbian Learning rule, and further he showed that his learning rule is simply an online update of Principal Component Analyzer. We present the details of this argument in the following paragraphs.

Starting from Equation 1 and following the same notation, Oja showed:

$$w_i^{t+1} = w_i^t + \eta x_i y$$

where t denotes the iteration. A straightforward way to avoid the exploding of weights is to apply normalization at the end of each iteration, yielding:

$$w_i^{t+1} = \frac{w_i^t + \eta x_i y}{(\sum_{i=1}^n (w_i^t + \eta x_i y)^2)^{\frac{1}{2}}}$$

where n denotes the number of neurons. The above equation can be further expanded into the following form:

$$w_i^{t+1} = \frac{w_i^t}{Z} + \eta(\frac{y x_i}{Z} + \frac{w_i \sum_j^n y x_j w_j}{Z^3}) + O(\eta^2)$$

where $Z = (\sum_i^n w_i^2)^{\frac{1}{2}}$. Further, two more assumptions are introduced: 1) η is small. Therefore $O(\eta^2)$ is approximately 0. 2) Weights are normalized, therefore $Z = (\sum_i^n w_i^2)^{\frac{1}{2}} = 1$.

When these two assumptions were introduced back to the previous equation, Oja's rule was proposed as following:

$$w_i^{t+1} = w_i^t + \eta y(x_i - y w_i^t) \tag{2}$$

Oja took a step further to show that a neuron that was updated with this rule was effectively performing Principal Component Analysis on the data. To show this, Oja first re-wrote Equation 2 as the following forms with two additional assumptions (Oja, 1982):

$$\frac{d}{d(t)} w_i^t = C w_i^t - ((w_i^t)^T C w_i^t) w_i^t$$

where C is the covariance matrix of input X. Then he proceeded to show this property with many conclusions from his another work (Oja and Karhunen, 1985) and linked back to PCA with the fact that components from PCA are eigenvectors and the first component is the eigenvector corresponding to largest eigenvalues of the covariance matrix. Intuitively, we could interpret this property with a simpler explanation: the eigenvectors of C are the solution when we maximize the rule updating function. Since w_i^t are the eigenvectors of the covariance matrix of X, we can get that w_i^t are the PCA.

Oja's learning rule concludes our story of learning rules of the early-stage neural network. Now we proceed to visit the ideas on neural models.

2.5 MCP Neural Model

While Donald Hebb is seen as the father of neural networks, the first model of neuron could trace back to six years ahead of the publication of Hebbian Learning Rule, when a neurophysiologist Warren McCulloch and a mathematician Walter Pitts speculated the inner workings of neurons and modeled a primitive neural network by electrical circuits based on their findings (McCulloch and Pitts, 1943). Their model, known as MCP neural model, was a linear step function upon weighted linearly interpolated data that could be described as:

$$y = \begin{cases} 1, \ \sum_i w_i x_i \geq \theta \quad \text{AND} \quad z_j = 0, \forall j \\ 0, \text{ otherwise} \end{cases}$$

where y stands for output, x_i stands for input of signals, w_i stands for the corresponding weights and z_j stands for the inhibitory input. θ stands for the threshold. The function is designed in a way that the activity of any inhibitory input completely prevents excitation of the neuron at any time.

Despite the resemblance between MCP Neural Model and modern perceptron, they are still different distinctly in many different aspects:

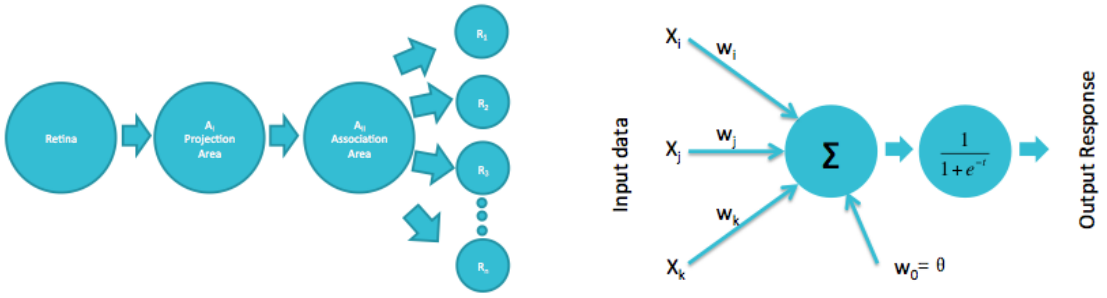
- MCP Neural Model is initially built as electrical circuits. Later we will see that the study of neural networks has borrowed many ideas from the field of electrical circuits.
- The weights of MCP Neural Model w_i are fixed, in contrast to the adjustable weights in modern perceptron. All the weights must be assigned with manual calculation.
- The idea of inhibitory input is quite unconventional even seen today. It might be an idea worth further study in modern deep learning research.

2.6 Perceptron

With the success of MCP Neural Model, Frank Rosenblatt further substantialized Hebbian Learning Rule with the introduction of perceptrons (Rosenblatt, 1958). While theorists like Hebb were focusing on the biological system in the natural environment, Rosenblatt constructed the electronic device named Perceptron that was showed with the ability to learn in accordance with associationism.

Rosenblatt (1958) introduced the perceptron with the context of the vision system, as showed in Figure 2(a). He introduced the rules of the organization of a perceptron as following:

- Stimuli impact on a retina of the sensory units, which respond in a manner that the pulse amplitude or frequency is proportional to the stimulus intensity.
- Impulses are transmitted to *Projection Area* (A_I). This projection area is optional.
- Impulses are then transmitted to *Association Area* through random connections. If the sum of impulse intensities is equal to or greater than the threshold (θ) of this unit, then this unit fires.



(a) Illustration of organization of a perceptron in (Rosenblatt, 1958)

(b) A typical perceptron in modern machine learning literature

Figure 2: Perceptrons: (a) A new figure of the illustration of organization of perceptron as in (Rosenblatt, 1958). (b) A typical perceptron nowadays, when A_I (Projection Area) is omitted.

- Response units work in the same fashion as those intermediate units.

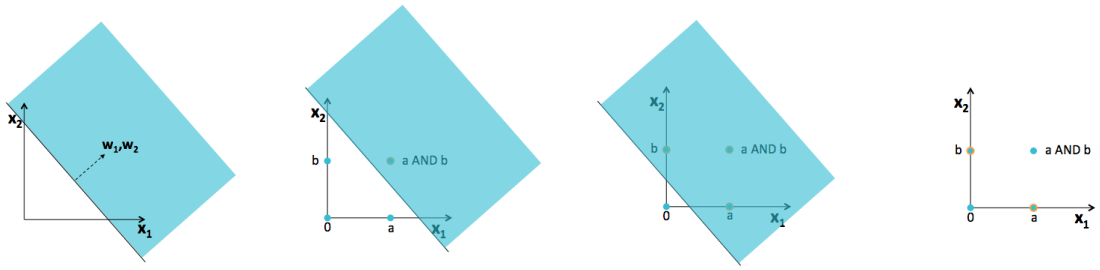
Figure 2(a) illustrates his explanation of perceptron. From left to right, the four units are sensory unit, projection unit, association unit and response unit respectively. Projection unit receives the information from sensory unit and passes onto association unit. This unit is often omitted in other description of similar models. With the omission of projection unit, the structure resembles the structure of nowadays perceptron in a neural network (as showed in Figure 2(b)): sensory units collect data, association units linearly adds these data with different weights and apply non-linear transform onto the thresholded sum, then pass the results to response units.

One distinction between the early stage neuron models and modern perceptrons is the introduction of non-linear activation functions (we use sigmoid function as an example in Figure 2(b)). This originates from the argument that linear threshold function should be softened to simulate biological neural networks (Bose et al., 1996) as well as from the consideration of the feasibility of computation to replace step function with a continuous one (Mitchell et al., 1997).

After Rosenblatt's introduction of Perceptron, Widrow et al. (1960) introduced a follow-up model called ADALINE. However, the difference between Rosenblatt's Perceptron and ADALINE is mainly on the algorithm aspect. As the primary focus of this paper is neural network models, we skip the discussion of ADALINE.

2.7 Perceptron's Linear Representation Power

A perceptron is fundamentally a linear function of input signals; therefore it is limited to represent linear decision boundaries like the logical operations like NOT, AND or OR, but not XOR when a more sophisticated decision boundary is required. This limitation was highlighted by Minski and Papert (1969), when they attacked the limitations of perceptions by emphasizing that perceptrons cannot solve functions like XOR or NXOR. As a result, very little research was done in this area until about the 1980s.



(a) (b) (c) (d)

Figure 3: The linear representation power of preceptron

To show a more concrete example, we introduce a linear preceptron with only two inputs x_1 and x_2, therefore, the decision boundary $w_1x_1 + w_2x_2$ forms a line in a two-dimensional space. The choice of threshold magnitude shifts the line horizontally and the sign of the function picks one side of the line as the halfspace the function represents. The halfspace is showed in Figure 3 (a).

In Figure 3 (b)-(d), we present two nodes a and b to denote to input, as well as the node to denote the situation when both of them trigger and a node to denote the situation when neither of them triggers. Figure 3 (b) and Figure 3 (c) show clearly that a linear perceptron can be used to describe AND and OR operation of these two inputs. However, in Figure 3 (d), when we are interested in XOR operation, the operation can no longer be described by a single linear decision boundary.

In the next section, we will show that the representation ability is greatly enlarged when we put perceptrons together to make a neural network. However, when we keep stacking one neural network upon the other to make a deep learning model, the representation power will not necessarily increase.

3. From Modern Neural Network to the Era of Deep Learning

In this section, we will introduce some important properties of neural networks. These properties partially explain the popularity neural network gains these days and also motivate the necessity of exploring deeper architecture. To be specific, we will discuss a set of universal approximation properties, in which each property has its condition. Then, we will show that although a shallow neural network is an universal approximator, deeper architecture can significantly reduce the requirement of resources while retaining the representation power. At last, we will also show some interesting properties discovered in the 1990s about backpropagation, which may inspire some related research today.

3.1 Universal Approximation Property

The step from perceptrons to basic neural networks is only placing the perceptrons together. By placing the perceptrons side by side, we get a single one-layer neural network and by stacking one one-layer neural network upon the other, we get a multi-layer neural network, which is often known as multi-layer perceptrons (MLP) (Kawaguchi, 2000).

One remarkable property of neural networks, widely known as universal approximation property, roughly describes that an MLP can represent any functions. Here we discussed this property in three different aspects:

- Boolean Approximation: an MLP of one hidden layer[1] can represent any boolean function exactly.
- Continuous Approximation: an MLP of one hidden layer can approximate any bounded continuous function with arbitrary accuracy.
- Arbitrary Approximation: an MLP of two hidden layers can approximate any function with arbitrary accuracy.

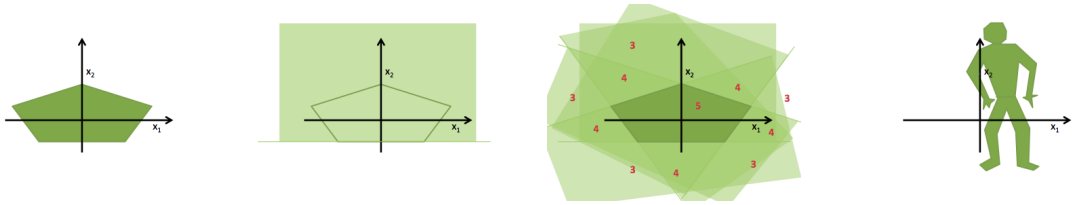
We will discuss these three properties in detail in the following paragraphs. To suit different readers' interest, we will first offer an intuitive explanation of these properties and then offer the proofs.

3.1.1 Representation of any Boolean Functions

This approximation property is very straightforward. In the previous section we have shown that every linear preceptron can perform either AND or OR. According to De Morgan's laws, every propositional formula can be converted into an equivalent Conjunctive Normal Form, which is an OR of multiple AND functions. Therefore, we simply rewrite the target Boolean function into an OR of multiple AND operations. Then we design the network in such a way: the input layer performs all AND operations, and the hidden layer is simply an OR operation.

The formal proof is not very different from this intuitive explanation, we skip it for simplicity.

1. Through this paper, we will follow the most widely accepted naming convention that calls a two-layer neural network as one hidden layer neural network.



(a) (b) (c) (d)

Figure 4: Example of Universal Approximation of any Bounded Continuous Functions

3.1.2 Approximation of any Bounded Continuous Functions

Continuing from the linear representation power of perceptron discussed previously, if we want to represent a more complex function, showed in Figure 4 (a), we can use a set of linear perceptrons, each of them describing a halfspace. One of these perceptrons is shown in Figure 4 (b), we will need five of these perceptrons. With these perceptrons, we can bound the target function out, as showed in Figure 4 (c). The numbers showed in Figure 4 (c) represent the number of subspaces described by perceptrons that fall into the corresponding region. As we can see, with an appropriate selection of the threshold (e.g. $\theta = 5$ in Figure 4 (c)), we can bound the target function out. Therefore, we can describe any bounded continuous function with only one hidden layer; even it is a shape as complicated as Figure 4 (d).

This property was first shown in (Cybenko, 1989) and (Hornik et al., 1989). To be specific, Cybenko (1989) showed that, if we have a function in the following form:

$$f(x) = \sum_i \omega_i \sigma(w_i^T x + \theta) \tag{3}$$

$f(x)$ is dense in the subspace of where it is in. In other words, for an arbitrary function $g(x)$ in the same subspace as $f(x)$, we have

$$|f(x) - g(x)| < \epsilon$$

where $\epsilon > 0$. In Equation 3, σ denotes the activation function (a squashing function back then), w_i denotes the weights for the input layer and ω_i denotes the weights for the hidden layer.

This conclusion was drawn with a proof by contradiction: With Hahn-Banach Theorem and Riesz Representation Theorem, the fact that the closure of $f(x)$ is not all the subspace where $f(x)$ is in contradicts the assumption that σ is an activation (squashing) function.

Till today, this property has drawn thousands of citations. Unfortunately, many of the later works cite this property inappropriately (Castro et al., 2000) because Equation 3 is not the widely accepted form of a one-hidden-layer neural network because it does not deliver a thresholded/squashed output, but a linear output instead. Ten years later after this property was shown, Castro et al. (2000) concluded this story by showing that when the final output is squashed, this universal approximation property still holds.

Note that, this property was shown with the context that activation functions are squashing functions. By definition, a squashing function $\sigma : R \to [0, 1]$ is a non-decreasing function

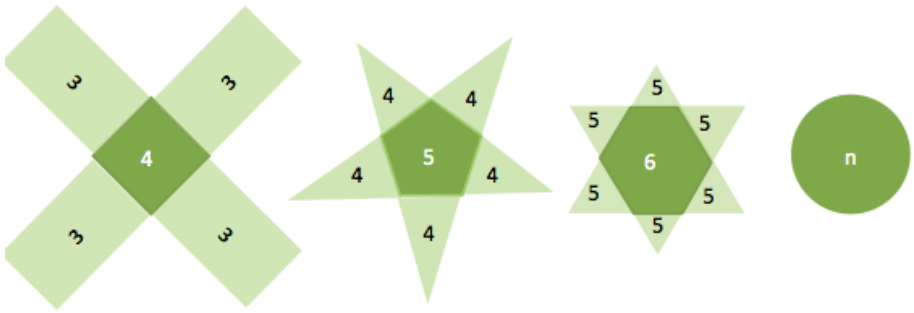


Figure 5: Threshold is not necessary with a large number of linear perceptrons.

with the properties $\lim_{x\to\infty}\sigma(x) = 1$ and $\lim_{x\to-\infty}\sigma(x) = 0$. Many activation functions of recent deep learning research do not fall into this category.

3.1.3 Approximation of Arbitrary Functions

Before we move on to explain this property, we need first to show a major property regarding combining linear perceptrons into neural networks. Figure 5 shows that as the number of linear perceptrons increases to bound the target function, the area outside the polygon with the sum close to the threshold shrinks. Following this trend, we can use a large number of perceptrons to bound a circle, and this can be achieved even without knowing the threshold because the area close to the threshold shrinks to nothing. What left outside the circle is, in fact, the area that sums to $\frac{N}{2}$, where N is the number of perceptrons used.

Therefore, a neural network with one hidden layer can represent a circle with arbitrary diameter. Further, we introduce another hidden layer that is used to combine the outputs of many different circles. This newly added hidden layer is only used to perform OR operation. Figure 6 shows an example that when the extra hidden layer is used to merge the circles from the previous layer, the neural network can be used to approximate any function. The target function is not necessarily continuous. However, each circle requires a large number of neurons, consequently, the entire function requires even more.

This property was showed in (Lapedes and Farber, 1988) and (Cybenko, 1988) respectively. Looking back at this property today, it is not arduous to build the connections between this property to Fourier series approximation, which, in informal words, states that every function curve can be decomposed into the sum of many simpler curves. With this linkage, to show this universal approximation property is to show that any one-hidden-layer neural network can represent one simple surface, then the second hidden layer sums up these simple surfaces to approximate an arbitrary function.

As we know, one hidden layer neural network simply performs a thresholded sum operation, therefore, the only step left is to show that the first hidden layer can represent a simple surface. To understand the "simple surface", with linkage to Fourier transform, one can imagine one cycle of the sinusoid for the one-dimensional case or a "bump" of a plane in the two-dimensional case.

Figure 6: How a neural network can be used to approximate a leaf shaped function.

For one dimension, to create a simple surface, we only need two sigmoid functions appropriately placed, for example, as following:

$$f_1(x) = \frac{h}{1 + e^{-(x+t_1)}}$$
$$f_2(x) = \frac{h}{1 + e^{x-t_2}}$$

Then, with $f_1(x) + f_2(x)$, we create a simple surface with height $2h$ from $t_1 \leq x \leq t_2$. This could be easily generalized to n-dimensional case, where we need $2n$ sigmoid functions (neurons) for each simple surface. Then for each simple surface that contributes to the final function, one neuron is added onto the second hidden layer. Therefore, despite the number of neurons need, one will never need a third hidden layer to approximate any function.

Similarly to how Gibbs phenomenon affects Fourier series approximation, this approximation cannot guarantee an exact representation.

The universal approximation properties showed a great potential of shallow neural networks at the price of exponentially many neurons at these layers. One followed-up question is that how to reduce the number of required neurons while maintaining the representation power. This question motivates people to proceed to deeper neural networks despite that shallow neural networks already have infinite modeling power. Another issue worth attention is that, although neural networks can approximate any functions, it is not trivial to find the set of parameters to explain the data. In the next two sections, we will discuss these two questions respectively.

3.2 The Necessity of Depth

The universal approximation properties of shallow neural networks come at a price of exponentially many neurons and therefore are not realistic. The question about how to maintain this expressive power of the network while reducing the number of computation units has been asked for years. Intuitively, Bengio and Delalleau (2011) suggested that it is nature to pursue deeper networks because 1) human neural system is a deep architecture (as we will see examples in Section 5 about human visual cortex.) and 2) humans tend to represent concepts at one level of abstraction as the composition of concepts at lower levels. Nowadays, the solution is to build deeper architectures, which comes from a conclusion that states the representation power of a k layer neural network with polynomial many neurons need to be expressed with exponentially many neurons if a $k-1$ layer structured is used. However, theoretically, this conclusion is still being completed.

This conclusion could trace back to three decades ago when Yao (1985) showed the limitations of shallow circuits functions. Hastad (1986) later showed this property with parity circuits: "there are functions computable in polynomial size and depth k but requires exponential size when depth is restricted to $k-1$". He showed this property mainly by the application of DeMorgan's law, which states that any AND or ORs can be rewritten as OR of ANDs and vice versa. Therefore, he simplified a circuit where ANDs and ORs appear one after the other by rewriting one layer of ANDs into ORs and therefore merge this operation to its neighboring layer of ORs. By repeating this procedure, he was able to represent the same function with fewer layers, but more computations.

Moving from circuits to neural networks, Delalleau and Bengio (2011) compared deep and shallow sum-product neural networks. They showed that a function that could be expressed with $O(n)$ neurons on a network of depth k required at least $O(2^{\sqrt{n}})$ and $O((n-1)^k)$ neurons on a two-layer neural network.

Further, Bianchini and Scarselli (2014) extended this study to a general neural network with many major activation functions including *tanh* and *sigmoid*. They derived the conclusion with the concept of Betti numbers, and used this number to describe the representation power of neural networks. They showed that for a shallow network, the representation power can only grow polynomially with respect to the number of neurons, but for deep architecture, the representation can grow exponentially with respect to the number of neurons. They also related their conclusion to VC-dimension of neural networks, which is $O(p^2)$ for *tanh* (Bartlett and Maass, 2003) where p is the number of parameters.

Recently, Eldan and Shamir (2015) presented a more thorough proof to show that depth of a neural network is exponentially more valuable than the width of a neural network, for a standard MLP with any popular activation functions. Their conclusion is drawn with only a few weak assumptions that constrain the activation functions to be mildly increasing, measurable, and able to allow shallow neural networks to approximate any univariate Lipschitz function. Finally, we have a well-grounded theory to support the fact that deeper network is preferred over shallow ones. However, in reality, many problems will arise if we keep increasing the layers. Among them, the increased difficulty of learning proper parameters is probably the most prominent one. Immediately in the next section, we will discuss the main drive of searching parameters for a neural network: Backpropagation.

3.3 Backpropagation and Its Properties

Before we proceed, we need to clarify that the name backpropagation, originally, is not referring to an algorithm that is used to learn the parameters of a neural network, instead, it stands for a technique that can help efficiently compute the gradient of parameters when gradient descent algorithm is applied to learn parameters (Hecht-Nielsen, 1989). However, nowadays it is widely recognized as the term to refer gradient descent algorithm with such a technique.

Compared to a standard gradient descent, which updates all the parameters with respect to error, backpropagation first propagates the error term at output layer back to the layer at which parameters need to be updated, then uses standard gradient descent to update parameters with respect to the propagated error. Intuitively, the derivation of backpropagation is about organizing the terms when the gradient is expressed with the chain rule. The derivation is neat but skipped in this paper due to the extensive resources available (Werbos, 1990; Mitchell et al., 1997; LeCun et al., 2015). Instead, we will discuss two interesting and seemingly contradictory properties of backpropagation.

3.3.1 Backpropagation Finds Global Optimal for Linear Separable Data

Gori and Tesi (1992) studied on the problem of local minima in backpropagation. Interestingly, when the society believes that neural networks or deep learning approaches are believed to suffer from local optimal, they proposed an architecture where global optimal is guaranteed. Only a few weak assumptions of the network are needed to reach global optimal, including

- Pyramidal Architecture: upper layers have fewer neurons
- Weight matrices are full row rank
- The number of input neurons cannot smaller than the classes/patterns of data.

However, their approaches may not be relevant anymore as they require the data to be linearly separable, under which condition that many other models can be applied.

3.3.2 Backpropagation Fails for Linear Separable Data

On the other hand, Brady et al. (1989) studied the situations when backpropagation fails on linearly separable data sets. He showed that there could be situations when the data is linearly separable, but a neural network learned with backpropagation cannot find that boundary. He also showed examples when this situation occurs.

His illustrative examples only hold when the misclassified data samples are significantly less than correctly classified data samples, in other words, the misclassified data samples might be just outliers. Therefore, this interesting property, when viewed today, is arguably a desirable property of backpropagation as we typically expect a machine learning model to neglect outliers. Therefore, this finding has not attracted many attentions.

However, no matter whether the data is an outlier or not, neural network should be able to overfit training data given sufficient training iterations and a legitimate learning algorithm, especially considering that Brady et al. (1989) showed that an inferior algorithm

was able to overfit the data. Therefore, this phenomenon should have played a critical role in the research of improving the optimization techniques. Recently, the studying of cost surfaces of neural networks have indicated the existence of saddle points (Choromanska et al., 2015; Dauphin et al., 2014; Pascanu et al., 2014), which may explain the findings of Brady *et al* back in the late 80s.

Backpropagation enables the optimization of deep neural networks. However, there is still a long way to go before we can optimize it well. Later in Section 8, we will briefly discuss more techniques related to the optimization of neural networks.

4. The Network as Memory and Deep Belief Nets

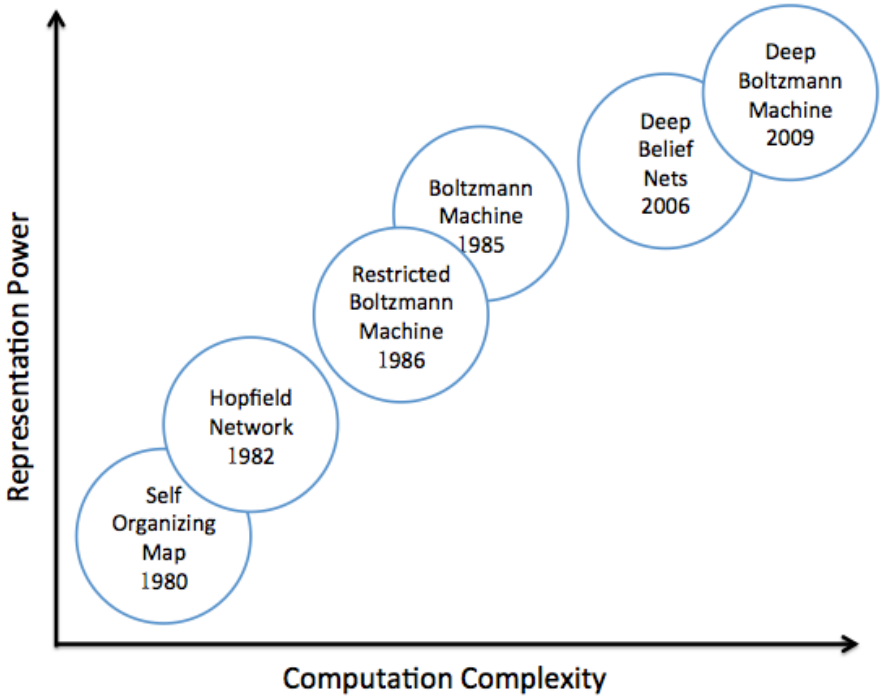


Figure 7: Trade off of representation power and computation complexity of several models, that guides the development of better models

With the background of how modern neural network is set up, we proceed to visit the each prominent branch of current deep learning family. Our first stop is the branch that leads to the popular Restricted Boltzmann Machines and Deep Belief Nets, and it starts as a model to understand the data unsupervisedly.

Figure 7 summarizes the model that will be covered in this Section. The horizontal axis stands for the computation complexity of these models while the vertical axis stands for the representation power. The six milestones that will be focused in this section are placed in the figure.

4.1 Self Organizing Map

The discussion starts with Self Organizing Map (SOM) invented by Kohonen (1990). SOM is a powerful technique that is primarily used in reducing the dimension of data, usually to one or two dimensions (Germano, 1999). While reducing the dimensionality, SOM also retains the topological similarity of data points. It can also be seen as a tool for clustering while imposing the topology on clustered representation. Figure 8 is an illustration of Self Organizing Map of two dimension hidden neurons. Therefore, it learns a two dimension representation of data. The upper shaded nodes denote the units of SOM that are used to

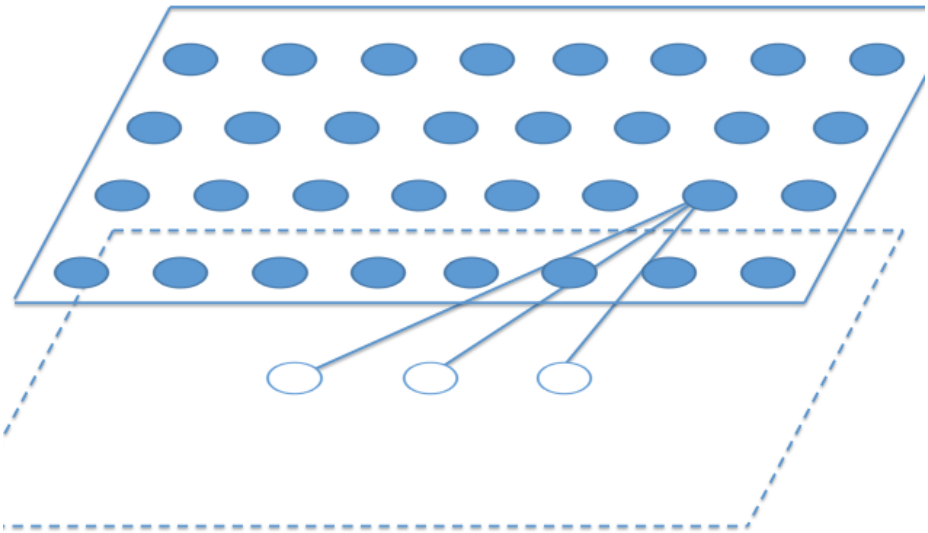


Figure 8: Illustration of Self-Organizing Map

represent data while the lower circles denote the data. There is no connection between the nodes in SOM [2].

The position of each node is fixed. The representation should not be viewed as only a numerical value. Instead, the position of it also matters. This property is different from some widely-accepted representation criterion. For example, we compare the case when one-hot vector and one-dimension SOM are used to denote colors: To denote *green* out of a set: $C = \{green, red, purple\}$, one-hot representation can use any vector of $(1, 0, 0)$, $(0, 1, 0)$ or $(0, 0, 1)$ as long as we specify the bit for *green* correspondingly. However, for a one-dimensional SOM, only two vectors are possible: $(1, 0, 0)$ or $(0, 0, 1)$. This is because that, since SOM aims to represent the data while retaining the similarity; and *red* and *purple* are much more similar than *green* and *red* or *green* and *purple*, *green* should not be represented in a way that it splits *red* and *purple*. One should notice that, this example is only used to demonstrate that the position of each unit in SOM matters. In practice, the values of SOM unit are not restricted to integers.

The learned SOM is usually a good tool for visualizing data. For example, if we conduct a survey on the happiness level and richness level of each country and feed the data into a two-dimensional SOM. Then the trained units should represent the happiest and richest country at one corner and represent the opposite country at the furthest corner. The rest two corners represent the richest, yet unhappiest and the poorest but happiest countries. The rest countries are positioned accordingly. The advantage of SOM is that it allows one

2. In some other literature, (Bullinaria, 2004) as an example, one may notice that there are connections in the illustrations of models. However, those connections are only used to represent the neighborhood relationship of nodes, and there is no information flowing via those connections. In this paper, as we will show many other models that rely on a clear illustration of information flow, we decide to save the connections to denote that.

to easily tell how a country is ranked among the world with a simple glance of the learned units (Guthikonda, 2005).

4.1.1 Learning Algorithm

With an understanding of the representation power of SOM, now we proceed to its parameter learning algorithm. The classic algorithm is heuristic and intuitive, as shown below: Here we use a two-dimensional SOM as example, and i, j are indexes of units; w is weight

---

Initialize weights of all units, $w_{i,j}$ $\forall$ i, j
for $t \leq N$ **do**
  Pick v_k randomly
  Select Best Matching Unit (BMU) as $p, q := \arg\min_{i,j} ||w_{ij} - v_k||_2^2$
  Select the nodes of interest as the neighbors of BMU. $I = \{w_{i,j} | \text{dist}(w_{i,j}, w_{p,q}) < r(t)\}$
  Update weights: $w_{i,j} = w_{i,j} + P(i, j, p, q)l(t)||w_{ij} - v_k||_2^2$, $\forall i, j \in I$
end for

---

of the unit; v denotes data vector; k is the index of data; t denotes the current iteration; N constrains the maximum number of steps allowed; $P(\cdot)$ denotes the penalty considering the distance between unit p, q and unit i, j; l is learning rate; r denotes a radius used to select neighbor nodes. Both l and r typically decrease as t increases. $|| \cdot ||_2^2$ denotes Euclidean distance and dist$(\cdot)$ denotes the distance on the position of units.

This algorithm explains how SOM can be used to learn a representation and how the similarities are retained as it always selects a subset of units that are similar with the data sampled and adjust the weights of units to match the data sampled.

However, this algorithm relies on a careful selection of the radius of neighbor selection and a good initialization of weights. Otherwise, although the learned weights will have a local property of topological similarity, it loses this property globally: sometimes, two similar clusters of similar events are separated by another dissimilar cluster of similar events. In simpler words, units of *green* may actually separate units of *red* and units of *purple* if the network is not appropriately trained. (Germano, 1999).

4.2 Hopfield Network

Hopfield Network is historically described as a form of recurrent[3] neural network, first introduced in (Hopfield, 1982). "Recurrent" in this context refers to the fact that the weights connecting the neurons are bidirectional. Hopfield Network is widely recognized because of its content-addressable memory property. This content-addressable memory property is a simulation of the spin glass theory. Therefore, we start the discussion from spin glass.

3. The term "recurrent" is very confusing nowadays because of the popularity recurrent neural network (RNN) gains.

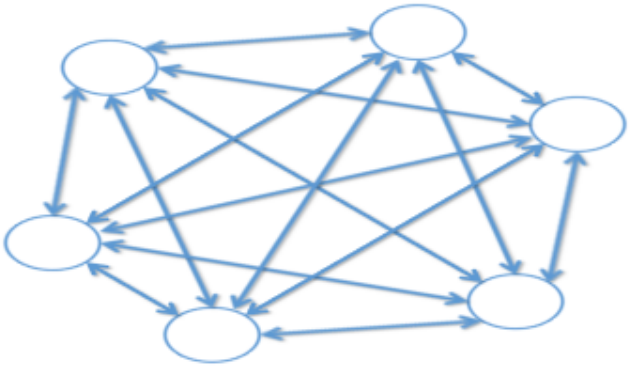


Figure 9: Illustration of Hopfield Network. It is a fully connected network of six binary thresholding neural units. Every unit is connected with data, therefore these units are denoted as unshaded nodes.

4.2.1 Spin Glass

The spin glass is physics term that is used to describe a magnetic phenomenon. Many works have been done for a detailed study of related theory (Edwards and Anderson, 1975; Mézard et al., 1990), so in this paper, we only describe this it intuitively.

When a group of dipoles is placed together in any space. Each dipole is forced to align itself with the field generated by these dipoles at its location. However, by aligning itself, it changes the field at other locations, leading other dipoles to flip, causing the field in the original location to change. Eventually, these changes will converge to a stable state.

To describe the stable state, we first define the total field at location j as

$$s_j = o_j + c^t \sum_k \frac{s_k}{d_{jk}^2}$$

where o_j is an external field, c^t is a constant that depends on temperature t, s_k is the polarity of the kth dipole and d_{jk} is the distance from location j to location k. Therefore, the total potential energy of the system is:

$$PE = \sum_j s_j o_j + c^t s_j \sum_k \frac{s_k}{d_{jk}^2} \tag{4}$$

The magnetic system will evolve until this potential energy is minimum.

4.2.2 Hopfield Network

Hopfield Network is a fully connected neural network with binary thresholding neural units. The values of these units are either 0 or 1[4]. These units are fully connected with bidirectional weights.

4. Some other literature may use -1 and 1 to denote the values of these units. While the choice of values does not affect the idea of Hopfiled Network, it changes the formulation of energy function. In this paper, we only discuss in the context of 0 and 1 as values.

With this setting, the energy of a Hopfield Network is defined as:

$$E = -\sum_i s_i b_i - \sum_{i,j} s_i s_j w_{i,j} \tag{5}$$

where s is the state of a unit, b denotes the bias; w denotes the bidirectional weights and i, j are indexes of units. This energy function closely connects to the potential energy function of spin glass, as showed in Equation 4.

Hopfield Network is typically applied to memorize the state of data. The weights of a network are designed or learned to make sure that the energy is minimized given the state of interest. Therefore, when another state presented to the network, while the weights are fixed, Hopfield Network can search for the states that minimize the energy and recover the state in memory. For example, in a face completion task, when some image of faces are presented to Hopfield Network (in a way that each unit of the network corresponds to each pixel of one image, and images are presented one after the other), the network can calculate the weights to minimize the energy given these faces. Later, if one image is corrupted or distorted and presented to this network again, the network is able to recover the original image by searching a configuration of states to minimize the energy starting from corrupted input presented.

The term "energy" may remind people of physics. To explain how Hopfield Network works in a physics scenario will be clearer: nature uses Hopfield Network to memorize the equilibrium position of a pendulum because, in an equilibrium position, the pendulum has the lowest gravitational potential energy. Therefore, whenever a pendulum is placed, it will converge back to the equilibrium position.

4.2.3 Learning and Inference

Learning of the weights of a Hopfield Network is straightforward (Gurney, 1997). The weights can be calculated as:

$$w_{i,j} = \sum_{i,j} (2s_i - 1)(2s_j - 1)$$

the notations are the same as Equation 5.

This learning procedure is simple, but still worth mentioning as it is an essential step of a Hopfield Network when it is applied to solve practical problems. However, we find that many online tutorials omit this step, and to make it worse, they refer the inference of states as learning/training. To remove the confusion, in this paper, similar to how terms are used in standard machine learning society, we refer the calculation of weights of a model (either from closed-form solution, or numerical solution) as "parameter learning" or "training". We refer the process of applying an existing model with weights known onto solving a real-world problem as "inference"[5] or "testing" (to decode a hidden state of data, e.g. to predict a label).

The inference of Hopfield Network is also intuitive. For a state of data, the network tests that if inverting the state of one unit, whether the energy will decrease. If so, the

5. "inference" is conventionally used in such a way in machine learning society, although some statisticians may disagree with this usage.

network will invert the state and proceed to test the next unit. This procedure is called **Asynchronous** update and this procedure is obviously subject to the sequential order of selection of units. A counterpart is known as **Synchronous** update when the network first tests for all the units and then inverts all the unit-to-invert simultaneously. Both of these methods may lead to a local optimal. Synchronous update may even result in an increasing of energy and may converge to an oscillation or loop of states.

4.2.4 Capacity

One distinct disadvantage of Hopfield Network is that it cannot keep the memory very efficient because a network of N units can only store memory up to $0.15N^2$ bits. While a network with N units has N^2 edges. In addition, after storing M memories (M instances of data), each connection has an integer value in range $[-M, M]$. Thus, the number of bits required to store N units are $N^2 log(2M + 1)$ (Hopfield, 1982). Therefore, we can safely draw the conclusion that although Hopfield Network is a remarkable idea that enables the network to memorize data, it is extremely inefficient in practice.

As follow-ups of the invention of Hopfield Network, many works are attempted to study and increase the capacity of original Hopfield Network (Storkey, 1997; Liou and Yuan, 1999; Liou and Lin, 2006). Despite these attempts made, Hopfield Network still gradually fades out of the society. It is replaced by other models that are inspired by it. Immediately following this section, we will discuss the popular Boltzmann Machine and Restricted Boltzmann Machine and study how these models are upgraded from the initial ideas of Hopfield Network and evolve to replace it.

4.3 Boltzmann Machine

Boltzmann Machine, invented by Ackley et al. (1985), is a stochastic with-hidden-unit version Hopfield Network. It got its name from Boltzmann Distribution.

4.3.1 Boltzmann Distribution

Boltzmann Distribution is named after Ludwig Boltzmann and investigated extensively by (Willard, 1902). It is originally used to describe the probability distribution of particles in a system over various possible states as following:

$$F(s) \propto e^{-\frac{E_s}{kT}}$$

where s stands for the state and E_s is the corresponding energy. k and T are Boltzmann's constant and thermodynamic temperature respectively. Naturally, the ratio of two distribution is only characterized by the difference of energies, as following:

$$r = \frac{F(s_1)}{F(s_2)} = e^{\frac{E_{s_2} - E_{s_1}}{kT}}$$

which is known as **Boltzmann factor**.

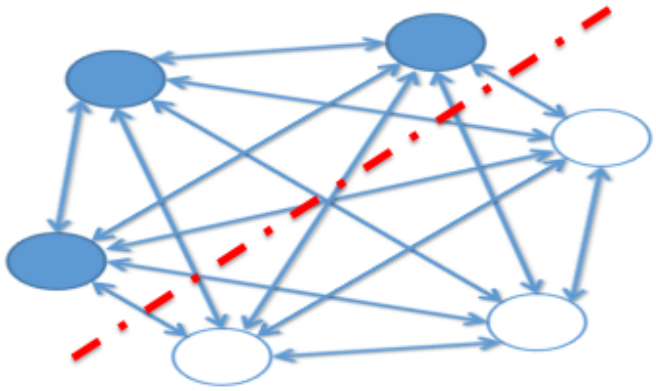


Figure 10: Illustration of Boltzmann Machine. With the introduction of hidden units (shaded nodes), the model conceptually splits into two parts: visible units and hidden units. The red dashed line is used to highlight the conceptual separation.

With how the distribution is specified by the energy, the probability is defined as the term of each state divided by a normalizer, as following:

$$p_{s_i} = \frac{e^{-\frac{E_{s_i}}{kT}}}{\sum_j e^{-\frac{E_{s_j}}{kT}}}$$

4.3.2 Boltzmann Machine

As we mentioned previously, Boltzmann Machine is a stochastic with-hidden-unit version Hopfield Network. Figure 10 introduces how the idea of hidden units is introduced that turns a Hopfield Network into a Boltzmann Machine. In a Boltzmann Machine, only visible units are connected with data and hidden units are used to assist visible units to describe the distribution of data. Therefore, the model conceptually splits into the visible part and hidden part, while it still maintains a fully connected network among these units.

"Stochastic" is introduced for Boltzmann Machine to be improved from Hopfield Network regarding leaping out of the local optimum or oscillation of states. Inspired by physics, a method to transfer state regardless current energy is introduced: Set a state to State 1 (which means the state is on) regardless of the current state with the following probability:

$$p = \frac{1}{1 + e^{-\frac{\Delta E}{T}}}$$

where ΔE stands for the difference of energies when the state is on and off, i.e. $\Delta E = E_{s=1} - E_{s=0}$. T stands for the temperature. The idea of T is inspired by a physics process that the higher the temperature is, the more likely the state will transfer[6]. In addition, the probability of higher energy state transferring to lower energy state will be always greater than the reverse process[7]. This idea is highly related to a very popular optimization

6. Molecules move faster when more kinetic energy is provided, which could be achieved by heating.
7. This corresponds to Zeroth Law of Thermodynamics.

algorithm called Simulated Annealing (Khachaturyan et al., 1979; Aarts and Korst, 1988) back then, but Simulated Annealing is hardly relevant to nowadays deep learning society. Regardless of the historical importance that the term T introduces, within this section, we will assume $T = 1$ as a constant, for the sake of simplification.

4.3.3 Energy of Boltzmann Machine

The energy function of Boltzmann Machine is defined the same as how Equation 5 is defined for Hopfield Network, except that now visible units and hidden units are noted separately, as following:

$$E(v, h) = -\sum_i v_i b_i - \sum_k h_k b_k - \sum_{i,j} v_i v_j w_{ij} - \sum_{i,k} v_i h_k w_{ik} - \sum_{k,l} h_k h_l w_{k,l}$$

where v stands for visible units, h stands for hidden units. This equation also connects back to Equation 4, except that Boltzmann Machine splits the energy function according to hidden units and visible units.

Based on this energy function, the probability of a joint configuration over both visible unit the hidden unit can be defined as following:

$$p(v, h) = \frac{e^{-E(v,h)}}{\sum_{m,n} e^{-E(m,n)}}$$

The probability of visible/hidden units can be achieved by marginalizing this joint probability.

For example, by marginalizing out hidden units, we can get the probability distribution of visible units:

$$p(v) = \frac{\sum_h e^{-E(v,h)}}{\sum_{m,n} e^{-E(m,n)}}$$

which could be used to sample visible units, i.e. generating data.

When Boltzmann Machine is trained to its stable state, which is called **thermal equilibrium**, the distribution of these probabilities $p(v, h)$ will remain constant because the distribution of energy will be a constant. However, the probability for each visible unit or hidden unit may vary and the energy may not be at their minimum. This is related to how thermal equilibrium is defined, where the only constant factor is the distribution of each part of the system.

Thermal equilibrium can be a hard concept to understand. One can imagine that pouring a cup of hot water into a bottle and then pouring a cup of cold water onto the hot water. At start, the bottle feels hot at bottom and feels cold at top and gradually the bottle feels mild as the cold water and hot water mix and heat is transferred. However, the temperature of the bottle becomes mild stably (corresponding to the distribution of $p(v, h)$) does not necessarily mean that the molecules cease to move (corresponding to each $p(v, h)$).

4.3.4 Parameter Learning

The common way to train the Boltzmann machine is to determine the parameters that maximize the likelihood of the observed data. Gradient descent on the log of the likelihood

function is usually performed to determine the parameters. For simplicity, the following derivation is based on a single observation.

First, we have the log likelihood function of visible units as

$$l(v;w) = \log p(v;w) = \log \sum_h e^{-E_{v,h}} - \log \sum_{m,n} e^{-E_{m,n}}$$

where the second term on RHS is the normalizer.

Now we take the derivative of log likelihood function w.r.t w, and simplify it, we have:

$$\begin{aligned}\frac{\partial l(v;w)}{\partial w} &= -\sum_h p(h|v)\frac{\partial E(v,h)}{\partial w} + \sum_{m,n} p(m,n)\frac{\partial E(m,n)}{\partial w} \\ &= -\mathbb{E}_{p(h|v)}\frac{\partial E(v,h)}{\partial w} + \mathbb{E}_{p(m,n)}\frac{\partial E(m,n)}{\partial w}\end{aligned}$$

where $\mathbb{E}$ denotes expectation. Thus the gradient of the likelihood function is composed of two parts. The first part is expected gradient of the energy function with respect to the conditional distribution $p(h|v)$. The second part is expected gradient of the energy function with respect to the joint distribution over all variable states. However, calculating these expectations is generally infeasible for any realistically-sized model, as it involves summing over a huge number of possible states/configurations. The general approach for solving this problem is to use Markov Chain Monte Carlo (MCMC) to approximate these sums:

$$\frac{\partial l(v;w)}{\partial w} = - < s_i, s_j >_{p(h_{data}|v_{data})} + < s_i, s_j >_{p(h_{model}|v_{model})} \tag{6}$$

where $< \cdot >$ denotes expectation.

Equation 6 is the difference between the expectation value of product of states while the data is fed into visible states and the expectation of product of states while no data is fed. The first term is calculated by taking the average value of the energy function gradient when the visible and hidden units are being driven by observed data samples. In practice, this first term is generally straightforward to calculate. Calculating the second term is generally more complicated and involves running a set of Markov chains until they reach the current models equilibrium distribution, then taking the average energy function gradient based on those samples.

However, this sampling procedure could be very computationally complicated, which motivates the topic in next section, the Restricted Boltzmann Machine.

4.4 Restricted Boltzmann Machine

Restricted Boltzmann Machine (RBM), originally known as Harmonium when invented by Smolensky (1986), is a version of Boltzmann Machine with a restriction that there is no connections either between visible units or between hidden units.

Figure 11 is an illustration of how Restricted Boltzmann Machine is achieved based on Boltzmann Machine (Figure 10): the connections between hidden units, as well as the connections between visible units are removed and the model becomes a bipartite graph. With this restriction introduced, the energy function of RBM is much simpler:

$$E(v,h) = -\sum_i v_i b_i - \sum_k h_k b_k - \sum_{i,k} v_i h_k w_{ik} \tag{7}$$

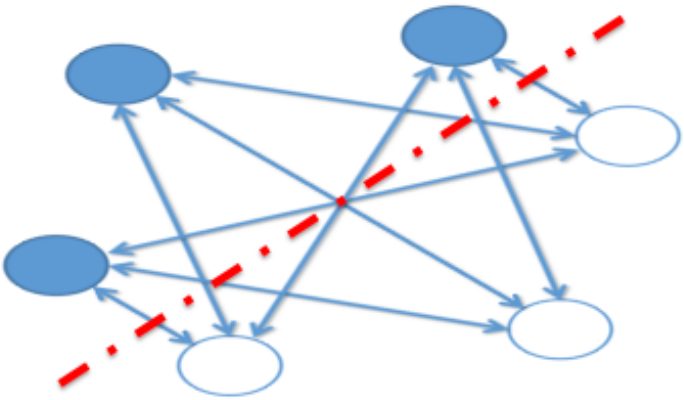


Figure 11: Illustration of Restricted Boltzmann Machine. With the restriction that there is no connections between hidden units (shaded nodes) and no connections between visible units (unshaded nodes), the Boltzmann Machine turns into a Restricted Boltzmann Machine. The model now is a a bipartite graph.

4.4.1 Contrastive Divergence

RBM can still be trained in the same way as how Boltzmann Machine is trained. Since the energy function of RBM is much simpler, the sampling method used to infer the second term in Equation 6 becomes easier. Despite this relative simplicity, this learning procedure still requires a large amount of sampling steps to approximate the model distribution.

To emphasize the difficulties of such a sampling mechanism, as well as to simplify follow-up introduction, we re-write Equation 6 with a different set of notations, as following:

$$\frac{\partial l(v;w)}{\partial w} = - < s_i, s_j >_{p_0} + < s_i, s_j >_{p_\infty} \tag{8}$$

here we use p_0 to denote data distribution and p_∞ to denote model distribution. Other notations remain unchanged. Therefore, the difficulty of mentioned methods to learn the parameters is that it requires potentially "infinitely" many sampling steps to approximate the model distribution.

Hinton (2002) overcame this issue magically, with the introduction of a method named Contrastive Divergence. Empirically, he found that one does not have to perform "infinitely" many sampling steps to converge to the model distribution, a finite k steps of sampling is enough. Therefore, Equation 8 is effectively re-written into:

$$\frac{\partial l(v;w)}{\partial w} = - < s_i, s_j >_{p_0} + < s_i, s_j >_{p_k}$$

Remarkably, Hinton (2002) showed that $k = 1$ is sufficient for the learning algorithm to work well in practice.

Carreira-Perpinan and Hinton (2005) attempted to justify Contrastive Divergence in theory, but their derivation led to a negative conclusion that Contrastive Divergence is a

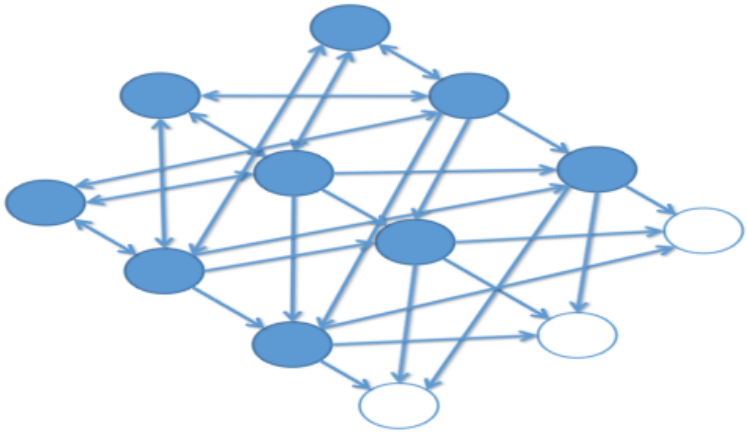


Figure 12: Illustration of Deep Belief Networks. Deep Belief Networks is not just stacking RBM together. The bottom layers (layers except the top one) do not have the bi-directional connections, but only connections top down.

biased algorithm, and a finite k cannot represent the model distribution. However, their empirical results suggested that finite k can approximate the model distribution well enough, resulting a small enough bias. In addition, the algorithm works well in practice, which strengthened the idea of Contrastive Divergence.

With the reasonable modeling power and a fast approximation algorithm, RBM quickly draws great attention and becomes one of the most fundamental building blocks of deep neural networks. In the following two sections, we will introduce two distinguished deep neural networks that are built based on RBM/Boltzmann Machine, namely Deep Belief Nets and Deep Boltzmann Machine.

4.5 Deep Belief Nets

Deep Belief Networks is introduced by Hinton et al. (2006)[8], when he showed that RBMs can be stacked and trained in a greedy manner.

Figure 12 shows the structure of a three-layer Deep Belief Networks. Different from stacking RBM, DBN only allows bi-directional connections (RBM-type connections) on the top one layer while the following bottom layers only have top-down connections. Probably a better way to understand DBN is to think it as multi-layer generative models. Despite the fact that DBN is generally described as a stacked RBM, it is quite different from putting one RBM on the top of the other. It is probably more appropriate to think DBN as a one-layer RBM with extended layers specially devoted to generating patterns of data.

Therefore, the model only needs to sample for the thermal equilibrium at the topmost layer and then pass the visible states top down to generate the data.

8. This paper is generally seen as the opening of nowadays Deep Learning era, as it first introduces the possibility of training a deep neural network by layerwise training

4.5.1 Parameter Learning

Parameter learning of a Deep Belief Network falls into two steps: the first step is layer-wise pre-training and the second step is fine-tuning.

Layerwise Pre-training The success of Deep Belief Network is largely due to the introduction of the layer-wised pretraining. The idea is simple, but the reason why it works still attracts researchers. The pre-training is simply to first train the network component by component bottom up: treating the first two layers as an RBM and train it, then treat the second layer and third layer as another RBM and train for the parameters.

Such an idea turns out to offer a critical support of the success of the later fine-tuning process. Several explanations have been attempted to explain the mechanism of pre-training:

- Intuitively, pre-training is a clever way of initialization. It puts the parameter values in the appropriate range for further fine-tuning.
- Bengio et al. (2007) suggested that unsupervised pre-training initializes the model to a point in parameter space which leads to a more effective optimization process, that the optimization can find a lower minimum of the empirical cost function.
- Erhan et al. (2010) empirically argued for a regularization explanation, that unsupervised pretraining guides the learning towards basins of attraction of minima that support better generalization from the training data set.

In addition to Deep Belief Networks, this pretraining mechanism also inspires the pre-training for many other classical models, including the autoencoders (Poultney et al., 2006; Bengio et al., 2007), Deep Boltzmann Machines (Salakhutdinov and Hinton, 2009) and some models inspired by these classical models like (Yu et al., 2010).

After the pre-training is performed, fine-tuning is carried out to further optimize the network to search for the parameters that lead to a lower minimum. For Deep Belief Networks, there are two different fine tuning strategies dependent on the goals of the network.

Fine Tuning for Generative Model Fine-tuning for a generative model is achieved with a contrastive version of wake-sleep algorithm (Hinton et al., 1995). This algorithm is intriguing for the reason that it is designed to interpret how the brain works. Scientists have found that sleeping is a critical process of brain function and it seems to be an inverse version of how we learn when we are awake. The wake-sleep algorithm also has two steps. In wake phase, we propagate information bottom up to adjust top-down weights for reconstructing the layer below. Sleep phase is the inverse of wake phase. We propagate the information top down to adjust bottom-up weights for reconstructing the layer above.

The contrastive version of this wake-sleep algorithm is that we add one Contrastive Divergence phase between wake phase and sleep phase. The wake phase only goes up to the visible layer of the top RBM, then we sample the top RBM with Contrastive Divergence, then a sleep phase starts from the visible layer of top RBM.

Fine Tuning for Discriminative Model The strategy for fine tuning a DBN as a discriminative model is to simply apply standard backpropagation to pre-trained model

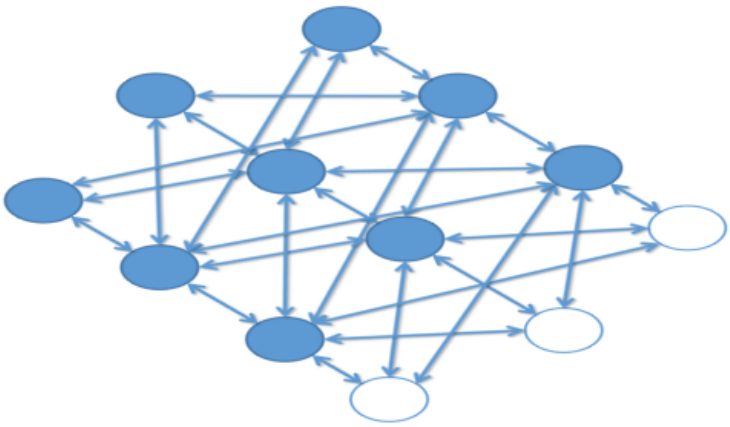


Figure 13: Illustration of Deep Boltzmann Machine. Deep Boltzmann Machine is more like stacking RBM together. Connections between every two layers are bidirectional.

since we have labels of data. However, pre-training is still necessary in spite of the generally good performance of backpropagation.

4.6 Deep Boltzmann Machine

The last milestone we introduce in the family of deep generative model is Deep Boltzmann Machine introduced by Salakhutdinov and Hinton (2009).

Figure 13 shows a three layer Deep Boltzmann Machine (DBM). The distinction between DBM and DBN mentioned in the previous section is that DBM allows bidirectional connections in the bottom layers. Therefore, DBM represents the idea of stacking RBMs in a much better way than DBN, although it might be clearer if DBM is named as Deep Restricted Boltzmann Machine.

Due to the nature of DBM, its energy function is defined as an extension of the energy function of an RBM (Equation 7), as showed in the following:

$$E(v,h) = -\sum_i v_i b_i - \sum_{n=1}^{N}\sum_k h_{n,k} b_{n,k} - \sum_{i,k} v_i w_{ik} h_k - \sum_{n=1}^{N-1}\sum_{k,l} h_{n,k} w_{n,k,l} h_{n+1,l}$$

for a DBM with N hidden layers.

This similarity of energy function grants the possibility of training DBM with constrative divergence. However, pre-training is typically necessary.

4.6.1 Deep Boltzmann Machine (DBM) v.s. Deep Belief Networks (DBN)

As their acronyms suggest, Deep Boltzmann Machine and Deep Belief Networks have many similarities, especially from the first glance. Both of them are deep neural networks originates from the idea of Restricted Boltzmann Machine. (The name "Deep Belief Network"

seems to indicate that it also partially originates from Bayesian Network (Krieg, 2001).) Both of them also rely on layerwise pre-training for a success of parameter learning.

However, the fundamental differences between these two models are dramatic, introduced by how the connections are made between bottom layers (un-directed/bi-directed v.s. directed). The bidirectional structure of DBM grants the possibility of DBM to learn a more complex pattern of data. It also grants the possibility for the approximate inference procedure to incorporate top-down feedback in addition to an initial bottom-up pass, allowing Deep Boltzmann Machines to better propagate uncertainty about ambiguous inputs.

4.7 Deep Generative Models: Now and the Future

Deep Boltzmann Machine is the last milestone we discuss in the history of generative models, but there are still much work after DBM and even more to be done in the future.

Lake et al. (2015) introduces a Bayesian Program Learning framework that can simulate human learning abilities with large scale visual concepts. In addition to its performance on one-shot learning classification task, their model passes the visual Turing Test in terms of generating handwritten characters from the worlds alphabets. In other words, the generative performance of their model is indistinguishable from human's behavior. Being not a deep neural model itself, their model outperforms several concurrent deep neural networks. Deep neural counterpart of the Bayesian Program Learning framework can be surely expected with even better performance.

Conditional image generation (given part of the image) is also another interesting topic recently. The problem is usually solved by Pixel Networks (Pixel CNN (van den Oord et al., 2016) and Pixel RNN (Oord et al., 2016)). However, given a part of the image seems to simplify the generation task.

Another prominent contribution to generative models is Generative Adversarial Networks. We will discuss it in Section 7.

5. Convolutional Neural Networks and Vision Problems

In this section, we will start to discuss a different family of models: the Convolutional Neural Network (CNN) family. Distinct from the family in the previous section, Convolutional Neural Network family mainly evolves from the knowledge of human visual cortex. Therefore, in this section, we will first introduce one of the most important reasons that account for the success of convolutional neural networks in vision problems: its bionic design to replicate human vision system. The nowadays convolutional neural networks probably originate more from the such a design rather than from the early-stage ancestors. With these background set-up, we will then briefly introduce the successful models that make themselves famous through the ImageNet Challenge (Deng et al., 2009). At last, we will present some known problems of the vision task that may guide the future research directions in vision tasks.

5.1 Visual Cortex

Convolutional Neural Network is widely known as being inspired by visual cortex, however, except that some publications discuss this inspiration briefly (Poggio and Serre, 2013; Cox and Dean, 2014), few resources present this inspiration thoroughly. In this section, we focus on the discussion about basics on visual cortex (Hubel and Wiesel, 1959), which lays the ground for further study in Convolutional Neural Networks.

The visual cortex of the brain, located in the occipital lobe which is located at the back of the skull, is a part of the cerebral cortex that plays an important role in processing visual information. Visual information coming from the eye, goes through a series of brain structures and reaches the visual cortex. The parts of the visual cortex that receive the sensory inputs is known as the primary visual cortex, also known as area V1. Visual information is further managed by extrastriate areas, including visual areas two (V2) and four (V4). There are also other visual areas (V3, V5, and V6), but in this paper, we primarily focus on the visual areas that are related to object recognition, which is known as ventral stream and consists of areas V1, V2, V4 and inferior temporal gyrus, which is one of the higher levels of the ventral stream of visual processing, associated with the representation of complex object features, such as global shape, like face perception (Haxby et al., 2000).

Figure 14 is an illustration of the ventral stream of the visual cortex. It shows the information process procedure from the retina which receives the image information and passes all the way to inferior temporal gyrus. For each component:

- Retina converts the light energy that comes from the rays bouncing off of an object into chemical energy. This chemical energy is then converted into action potentials that are transferred onto primary visual cortex. (In fact, there are several other brain structures involved between retina and V1, but we omit these structures for simplicity[9].)

9. We deliberately discuss the components that have connections with established technologies in convolutional neural network, one who is interested in developing more powerful models is encouraged to investigate other components.

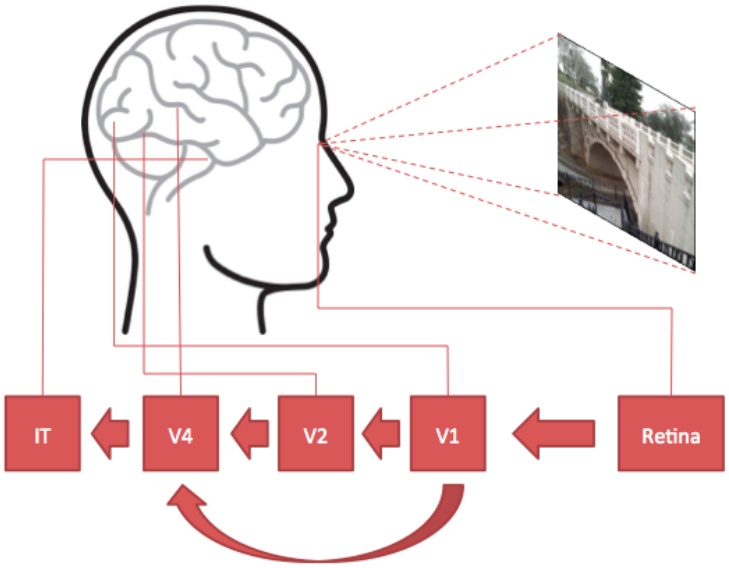


Figure 14: A brief illustration of ventral stream of the visual cortex in human vision system. It consists of primary visual cortex (V1), visual areas (V2 and V4) and inferior temporal gyrus.

- Primary visual cortex (V1) mainly fulfills the task of edge detection, where an edge is an area with strongest local contrast in the visual signals.

- V2, also known as secondary visual cortex, is the first region within the visual association area. It receives strong feedforward connections from V1 and sends strong connections to later areas. In V2, cells are tuned to extract mainly simple properties of the visual signals such as orientation, spatial frequency, and colour, and a few more complex properties.

- V4 fulfills the functions including detecting object features of intermediate complexity, like simple geometric shapes, in addition to orientation, spatial frequency, and color. V4 is also shown with strong attentional modulation (Moran and Desimone, 1985). V4 also receives direct input from V1.

- Inferior temporal gyrus (TI) is responsible for identifying the object based on the color and form of the object and comparing that processed information to stored memories of objects to identify that object (Kolb et al., 2014). In other words, IT performs the semantic level tasks, like face recognition.

Many of the descriptions of functions about visual cortex should revive a recollection of convolutional neural networks for the readers that have been exposed to some relevant technical literature. Later in this section, we will discuss more details about convolutional neural networks, which will help build explicit connections. Even for readers that barely

have knowledge in convolutional neural networks, this hierarchical structure of visual cortex should immediately ring a bell about neural networks.

Besides convolutional neural networks, visual cortex has been inspiring the works in computer vision for a long time. For example, Li (1998) built a neural model inspired by the primary visual cortex (V1). In another granularity, Serre et al. (2005) introduced a system with feature detections inspired from the visual cortex. De Ladurantaye et al. (2012) published a book describing the models of information processing in the visual cortex. Poggio and Serre (2013) conducted a more comprehensive survey on the relevant topic, but they didn't focus on any particular subject in detail in their survey. In this section, we discuss the connections between visual cortex and convolutional neural networks in details. We will begin with Neocogitron, which borrows some ideas from visual cortex and later inspires convolutional neural network.

5.2 Neocogitron and Visual Cortex

Neocogitron, proposed by Fukushima (1980), is generally seen as the model that inspires Convolutional Neural Networks on the computation side. It is a neural network that consists of two different kinds of layers (S-layer as feature extractor and C-layer as structured connections to organize the extracted features.)

S-layer consists of a number of S-cells that are inspired by the cell in primary visual cortex. It serves as a feature extractor. Each S-cell can be ideally trained to be responsive to a particular feature presented in its receptive field. Generally, local features such as edges in particular orientations are extracted in lower layers while global features are extracted in higher layers. This structure highly resembles how human conceive objects. C-layer resembles complex cell in the higher pathway of visual cortex. It is mainly introduced for shift invariant property of features extracted by S-layer.

5.2.1 Parameter Learning

During parameter learning process, only the parameters of S-layer are updated. Neocogitron can also be trained unsupervisedly, for a good feature extractor out of S-layers. The training process for S-layer is very similar to Hebbian Learning rule, which strengthens the connections between S-layer and C-layer for whichever S-cell shows the strongest response. This training mechanism also introduces the problem Hebbian Learning rule introduces, that the strength of connections will saturate (since it keeps increasing). The solution was also introduced by Fukushima (1980), which was introduced with the name "inhibitory cell". It performed the function as a normalization to avoid the problem.

5.3 Convolutional Neural Network and Visual Cortex

Now we proceed from Neocogitron to Convolutional Neural Network. First, we will introduce the building components: convolutional layer and subsampling layer. Then we assemble these components to present Convolutional Neural Network, using LeNet as an example.

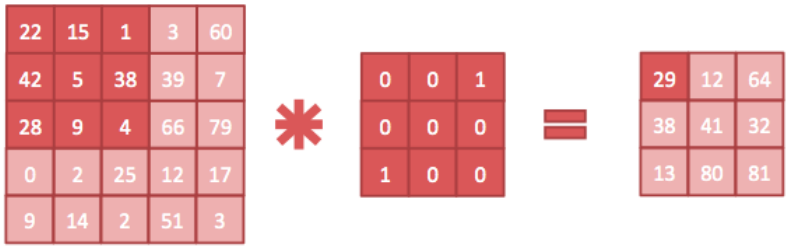


Figure 15: A simple illustration of two dimension convolution operation.

5.3.1 Convolution Operation

Convolution operation is strictly just a mathematical operation, which should be treated equally with other operations like addition or multiplication and should not be discussed particularly in a machine learning literature. However, we still discuss it here for completeness and for the readers who may not be familiar with it.

Convolution is a mathematical operation on two functions (e.g. f and g) and produces a third function h, which is an integral that expresses the amount of overlap of one function (f) as it is shifted over the other function (g). It is described formally as the following:

$$h(t) = \int_{-\infty}^{\infty} f(\tau)g(t-\tau)d\tau$$

and denoted as $h = f \star g$.

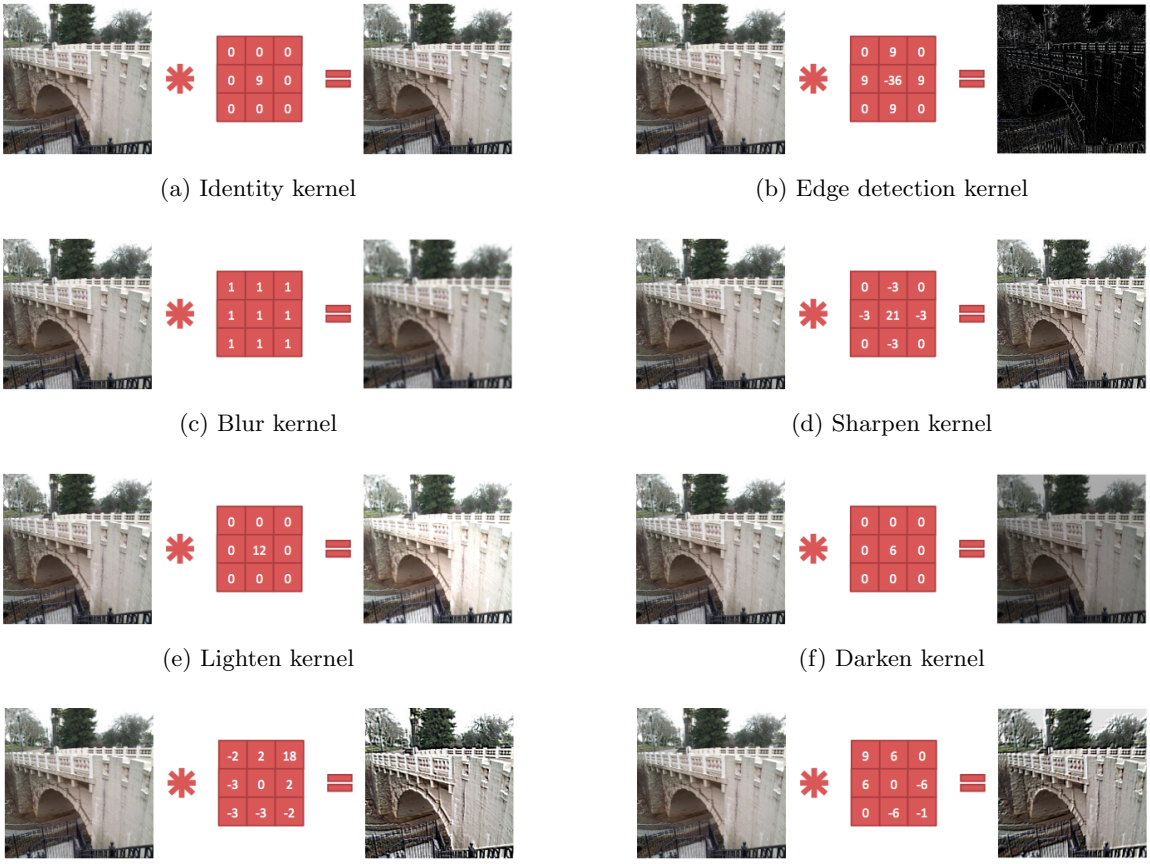
Convolutional neural network typically works with two-dimensional convolution operation, which could be summarized in Figure 15.

As showed in Figure 15, the leftmost matrix is the input matrix. The middle one is usually called a kernel matrix. Convolution is applied to these matrices and the result is showed as the rightmost matrix. The convolution process is an element-wise product followed by a sum, as showed in the example. When the left upper 3×3 matrix is convoluted with the kernel, the result is 29. Then we slide the target 3×3 matrix one column right, convoluted with the kernel and get the result 12. We keep sliding and record the results as a matrix. Because the kernel is 3×3, every target matrix is 3×3, thus, every 3×3 matrix is convoluted to one digit and the whole 5×5 matrix is shrunk into 3×3 matrix. (Because $5 - (3 - 1) = 3$. The first 3 means the size of the kernel matrix.)

One should realize that convolution is locally shift invariant, which means that for many different combinations of how the nine numbers in the upper 3×3 matrix are placed, the convoluted result will be 29. This invariant property plays a critical role in vision problem because that in an ideal case, the recognition result should not be changed due to shift or rotation of features. This critical property is used to be solved elegantly by Lowe (1999); Bay et al. (2006), but convolutional neural network brought the performance up to a new level.

5.3.2 Connection between CNN and Visual Cortex

With the ideas about two dimension convolution, we further discuss how convolution is a useful operation that can simulate the tasks performed by visual cortex.



(a) Identity kernel

(b) Edge detection kernel

(c) Blur kernel

(d) Sharpen kernel

(e) Lighten kernel

(f) Darken kernel

(g) Random kernel 1

(h) Random kernel 2

Figure 16: Convolutional kernels example. Different kernels applied to the same image will result in differently processed images. Note that there is a $\frac{1}{9}$ divisor applied to these kernels.

The convolution operation is usually known as kernels. By different choices of kernels, different operations of the images could be achieved. Operations are typically including identity, edge detection, blur, sharpening etc. By introducing random matrices as convolution operator, some interesting properties might be discovered.

Figure 16 is an illustration of some example kernels that are applied to the same figure. One can see that different kernels can be applied to fulfill different tasks. Random kernels can also be applied to transform the image into some interesting outcomes.

Figure 16 (b) shows that edge detection, which is one of the central tasks of primary visual cortex, can be fulfilled by a clever choice of kernels. Furthermore, clever selection of kernels can lead us to a success replication of visual cortex. As a result, learning a meaningful convolutional kernel (i.e. parameter learning) is one of the central tasks in convolutional neural networks when applied to vision tasks. This also explains that why

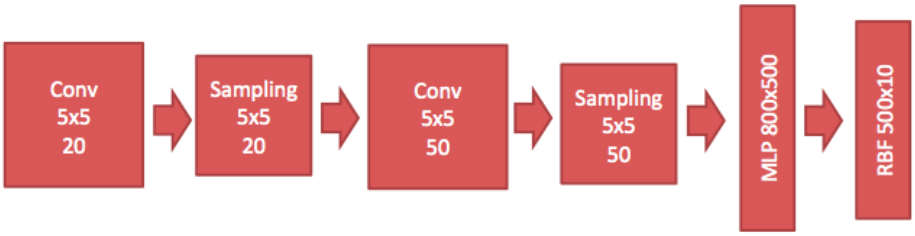


Figure 17: An illustration of LeNet, where Conv stands for convolutional layer and Sampling stands for SubSampling Layer.

many well-trained popular models can usually perform well in other tasks with only limited fine-tuning process: the kernels have been well trained and can be universally applicable.

With the understanding of the essential role convolution operation plays in vision tasks, we proceed to investigate some major milestones along the way.

5.4 The Pioneer of Convolutional Neural Networks: LeNet

This section is devoted to a model that is widely recognized as the first convolutional neural network: LeNet, invented by Le Cun et al. (1990) (further made popular with (LeCun et al., 1998a)). It is inspired from the Neocogitron. In this section, we will introduce convolutional neural network via introducing LeNet.

Figure 17 shows an illustration of the architecture of LeNet. It consists of two pairs of Convolutional Layer and Subsampling Layer and is further connected with fully connected layer and an RBF layer for classification.

5.4.1 Convolutional Layer

A convolutional layer is primarily a layer that performs convolution operation. As we have discussed previously, a clever selection of convolution kernel can effectively simulate the task of visual cortex. Convolutional layer introduces another operation after convolution to assist the simulation to be more successful: the non-linearity transform.

Considering a ReLU (Nair and Hinton, 2010) non-linearity transform, which is defined as following:

$$f(x) = \max(0, x)$$

which is a transform that removes the negative part of the input, resulting in a clearer contrast of meaningful features as opposed to other side product the kernel produces.

Therefore, this non-linearity grants the convolution more power in extracting useful features and allows it to simulate the functions of visual cortex more closely.

5.4.2 Subsampling Layer

Subsampling Layer performs a simpler task. It only samples one input out every region it looks into. Some different strategies of sampling can be considered, like max-pooling (taking the maximum value of the input), average-pooling (taking the averaged value of input) or even probabilistic pooling (taking a random one.) (Lee et al., 2009).

Sampling turns the input representations into smaller and more manageable embeddings. More importantly, sampling makes the network invariant to small transformations, distortions, and translations in the input image. A small distortion in the input will not change the outcome of pooling since we take the maximum/average value in a local neighborhood.

5.4.3 LeNet

With the two most important components introduced, we can stack them together to assemble a convolutional neural network. Following the recipe of Figure 17, we will end up with the famous LeNet.

LeNet is known as its ability to classify digits and can handle a variety of different problems of digits including variances in position and scale, rotation and squeezing of digits, and even different stroke width of the digit. Meanwhile, with the introduction of LeNet, LeCun et al. (1998b) also introduces the MNIST database, which later becomes the standard benchmark in digit recognition field.

5.5 Milestones in ImageNet Challenge

With the success of LeNet, Convolutional Neural Network has been shown with great potential in solving vision tasks. These potentials have attracted a large number of researchers aiming to solve vision task regarding object recognition in CIFAR classification (Krizhevsky and Hinton, 2009) and ImageNet challenge (Russakovsky et al., 2015). Along with this path, several superstar milestones have attracted great attentions and has been applied to other fields with good performance. In this section, we will briefly discuss these models.

5.5.1 AlexNet

While LeNet is the one that starts the era of convolutional neural networks, AlexNet, invented by Krizhevsky et al. (2012), is the one that starts the era of CNN used for ImageNet classification. AlexNet is the first evidence that CNN can perform well on this historically difficult ImageNet dataset and it performs so well that leads the society into a competition of developing CNNs.

The success of AlexNet is not only due to this unique design of architecture but also due to the clever mechanism of training. To avoid the computationally expensive training process, AlexNet has been split into two streams and trained on two GPUs. It also used data augmentation techniques that consist of image translations, horizontal reflections, and patch extractions.

The recipe of AlexNet is shown in Figure 18. However, rarely any lessons can be learned from the architecture of AlexNet despite its remarkable performance. Even more unfortunately, the fact that this particular architecture of AlexNet does not have a well-grounded theoretical support pushes many researchers to blindly burn computing resources

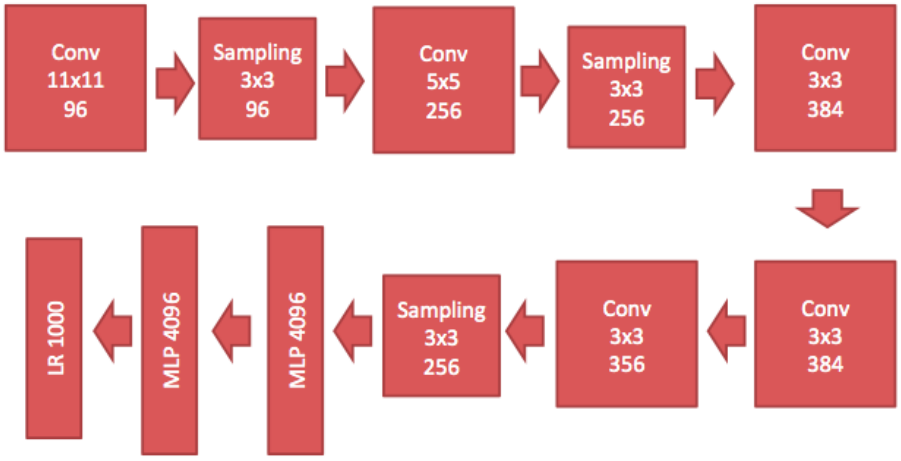


Figure 18: An illustration of AlexNet

to test for a new architecture. Many models have been introduced during this period, but only a few may be worth mentioning in the future.

5.5.2 VGG

In the blind competition of exploring different architectures, Simonyan and Zisserman (2014) showed that simplicity is a promising direction with a model named VGG. Although VGG is deeper (19 layer) than other models around that time, the architecture is extremely simplified: all the layers are 3×3 convolutional layer with a 2×2 pooling layer. This simple usage of convolutional layer simulates a larger filter while keeping the benefits of smaller filter sizes, because the combination of two 3×3 convolutional layers has an effective receptive field of a 5×5 convolutional layer, but with fewer parameters.

The spatial size of the input volumes at each layer will decrease as a result of the convolutional and pooling layers, but the depth of the volumes increases because of the increased number of filters (in VGG, the number of filters doubles after each pooling layer). This behavior reinforces the idea of VGG to shrink spatial dimensions, but grow depth.

VGG is not the winner of the ImageNet competition of that year (The winner is GoogLeNet invented by Szegedy et al. (2015)). GoogLeNet introduced several important concepts like Inception module and the concept later used by R-CNN (Girshick et al., 2014; Girshick, 2015; Ren et al., 2015), but the arbitrary/creative design of architecture barely contribute more than what VGG does to the society, especially considering that Residual Net, following the path of VGG, won the ImageNet challenge in an unprecedented level.

5.5.3 Residual Net

Residual Net (ResNet) is a 152 layer network, which was ten times deeper than what was usually seen during the time when it was invented by He et al. (2015). Following the path VGG introduces, ResNet explores deeper structure with simple layer. However, naively

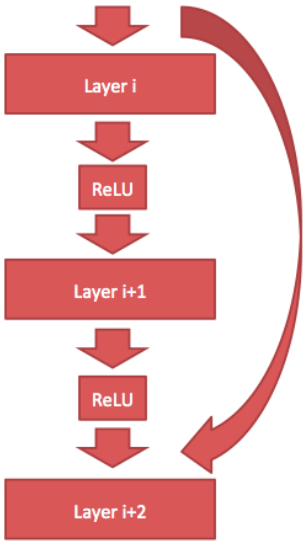


Figure 19: An illustration of Residual Block of ResNet

increasing the number of layers will only result in worse results, for both training cases and testing cases (He et al., 2015).

The breakthrough ResNet introduces, which allows ResNet to be substantially deeper than previous networks, is called Residual Block. The idea behind a Residual Block is that some input of a certain layer (denoted as x) can be passed to the component two layers later either following the traditional path which involves convolutional layers and ReLU transform succession (we denote the result as $f(x)$), or going through an express way that directly passes x there. As a result, the input to the component two layers later is $f(x) + x$ instead of what is typically seen as $f(x)$. The idea of Residual Block is illustrated in Figure 19.

In a complementary work, He et al. (2016) validated that residual blocks are essential for propagating information smoothly, therefore simplifies the optimization. They also extended the ResNet to a 1000-layer version with success on CIFAR data set.

Another interesting perspective of ResNet is provided by (Veit et al., 2016). They showed that ResNet behave behaves like ensemble of shallow networks: the express way introduced allows ResNet to perform as a collection of independent networks, each network is significantly shallower than the integrated ResNet itself. This also explains why gradient can be passed through the ultra-deep architecture without being vanished. (We will talk more about vanishing gradient problem when we discuss recurrent neural network in the next section.) Another work, which is not directly relevant to ResNet, but may help to understand it, is conducted by Hariharan et al. (2015). They showed that features from lower layers are informative in addition to what can be summarized from the final layer.

ResNet is still not completely vacant from clever designs. The number of layers in the whole network and the number of layers that Residual Block allows identity to bypass are still choices that require experimental validations. Nonetheless, to a great extent, ResNet has shown that critical reasoning can help the development of CNN better than blind

experimental trails. In addition, the idea of Residual Block has been found in the actual visual cortex (In the ventral stream of the visual cortex, V4 can directly accept signals from primary visual cortex).

With the introduction of these state-of-the-art neural models that are successful in these challenges, Canziani et al. (2016) conducted a comprehensive experimental study comparing these models. Upon comparison, they showed that there is still room for improvement on fully connected layers that show strong inefficiencies for smaller batches of images.

5.6 Challenges and Chances for Fundamental Vision Problems

ResNet is not the end of the story. New models and techniques appear every day to push the limit of CNNs further. For example, Zhang et al. (2016b) took a step further and put Residual Block inside Residual Block. Zagoruyko and Komodakis (2016) attempted to decrease the depth of network by increasing the width. However, incremental works of this kind are not in the scope of this paper.

We would like to end the story of Convolutional Neural Networks with some of the current challenges of fundamental vision problems that may not able to be solved naively by investigation of machine learning techniques.

5.6.1 Network Property and Vision Blindness Spot

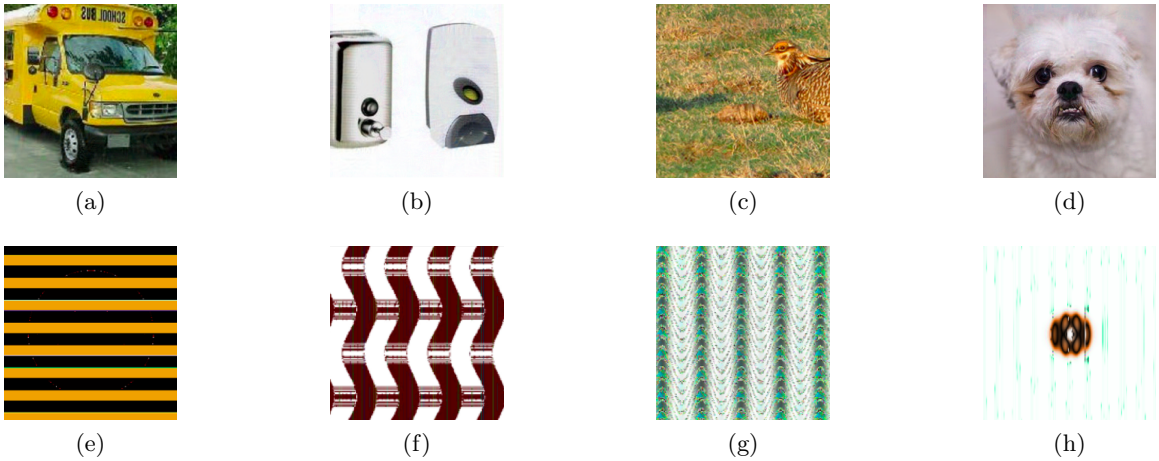
Convolutional Neural Networks have reached to an unprecedented accuracy in object detection. However, it may still be far from industry reliable application due to some intriguing properties found by Szegedy et al. (2013).

Szegedy et al. (2013) showed that they could force a deep learning model to misclassify an image simply by adding perturbations to that image. More importantly, these perturbations may not even be observed by naked human eyes. In other words, two objects that look almost the same to human, may be recognized as different objects by a well-trained neural network (for example, AlexNet). They have also shown that this property is more likely to be a modeling problem, in contrast to problems raised by insufficient training.

On the other hand, Nguyen et al. (2015) showed that they could generate patterns that convey almost no information to human, being recognized as some objects by neural networks with high confidence (sometimes more than 99%). Since neural networks are typically forced to make a prediction, it is not surprising to see a network classify a meaningless patter into something, however, this high confidence may indicate that the fundamental differences between how neural networks and human learn to know this world.

Figure 20 shows some examples from the aforementioned two works. With construction, we can show that the neural networks may misclassify an object, which should be easily recognized by the human, to something unusual. On the other hand, a neural network may also classify some weird patterns, which are not believed to be objects by the human, to something we are familiar with. Both of these properties may restrict the usage of deep learning to real world applications when a reliable prediction is necessary.

Even without these examples, one may also realize that the reliable prediction of neural networks could be an issue due to the fundamental property of a matrix: the existence of null space. As long as the perturbation happens within the null space of a matrix, one may be able to alter an image dramatically while the neural network still makes the



(a) (b) (c) (d)

(e) (f) (g) (h)

Figure 20: Illustrations of some mistakes of neural networks. (a)-(d) (from (Szegedy et al., 2013)) are adversarial images that are generated based on original images. The differences between these and the original ones are un-observable by naked eye, but the neural network can successfully classify original ones but fail adversarial ones. (e)-(h) (from (Nguyen et al., 2015)) are patterns that are generated. A neural network classify them into (e) school bus, (f) guitar, (g) peacock and (h) Pekinese respectively.

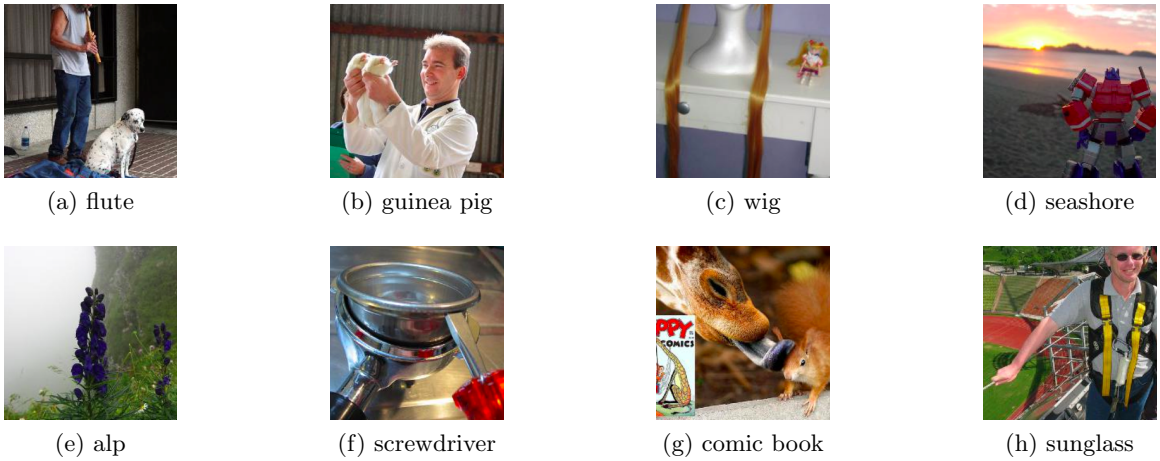
misclassification with high confidence. Null space works like a blind spot to a matrix and changes within null space are never sensible to the corresponding matrix.

This blind spot should not discourage the promising future of neural networks. On the contrary, it makes the convolutional neural network resemble the human vision system in a deeper level. In the human vision system, blind spots (Gregory and Cavanagh, 2011) also exist (Wandell, 1995). Interesting work might be seen about linking the flaws of human vision system to the defects of neural networks and helping to overcome these defects in the future.

5.6.2 Human Labeling Preference

At the very last, we present some of the misclassified images of ResNet on ImageNet Challenge. Hopefully, some of these examples could inspire some new methodologies invented for the fundamental vision problem.

Figure 21 shows some misclassified images of ResNet when applied to ImageNet Challenge. These labels, provided by human effort, are very unexpected even to many other humans. Therefore, the 3.6% error rate of ResNet (a general human usually predicts with error rate 5%-10%) is probably hitting the limit since the labeling preference of an annotator is harder to predict than the actual labels. For example, Figure 21 (a),(b),(h) are labeled as a tiny part of the image, while there are more important contents expressed by the image. On the other hand, Figure 21 (d) (e) are annotated as the background of the image while that image is obviously centering on other object.



(a) flute (b) guinea pig (c) wig (d) seashore

(e) alp (f) screwdriver (g) comic book (h) sunglass

Figure 21: Some failed images of ImageNet classification by ResNet and the primary label associated with the image.

To further improve the performance ResNet reached, one direction might be to modeling the annotators' labeling preference. One assumption could be that annotators prefer to label an image to make it distinguishable. Some established work to modeling human factors (Wilson et al., 2015) could be helpful.

However, the more important question is that whether it is worth optimizing the model to increase the testing results on ImageNet dataset, since remaining misclassifications may not be a result of the incompetency of the model, but problems of annotations.

The introduction of other data sets, like COCO (Lin et al., 2014), Flickr (Plummer et al., 2015), and VisualGenome (Krishna et al., 2016) may open a new era of vision problems with more competitive challenges. However, the fundamental problems and experiences that this section introduces should never be forgotten.

6. Time Series Data and Recurrent Networks

In this section, we will start to discuss a new family of deep learning models that have attracted many attentions, especially for the tasks on time series data, or sequential data. The Recurrent Neural Network (RNN) is a class of neural network whose connections of units form a directed cycle; this nature grants its ability to work with temporal data. It has also been discussed in literature like (Grossberg, 2013) and (Lipton et al., 2015). In this paper, we will continue to offer complementary views to other surveys with an emphasis on the evolutionary history of the milestone models and aim to provide insights into the future direction of coming models.

6.1 Recurrent Neural Network: Jordan Network and Elman Network

As we have discussed previously, Hopfield Network is widely recognized as a recurrent neural network, although its formalization is distinctly different from how recurrent neural network is defined nowadays. Therefore, despite that other literature tend to begin the discussion of RNN with Hopfield Network, we will not treat it as a member of RNN family to avoid unnecessary confusion.

The modern definition of "recurrent" is initially introduced by Jordan (1986) as:

> If a network has one or more cycles, that is, if it is possible to follow a path from a unit back to itself, then the network is referred to as *recurrent*. A *nonrecurrent* network has no cycles.

His model in (Jordan, 1986) is later referred to as Jordan Network. For a simple neural network with one hidden layer, with input denoted as X, weights of hidden layer denoted as w_h and weights of output layer denoted as w_y, weights of recurrent computation denoted as w_r, hidden representation denoted as h and output denoted as y, Jordan Network can be formulated as

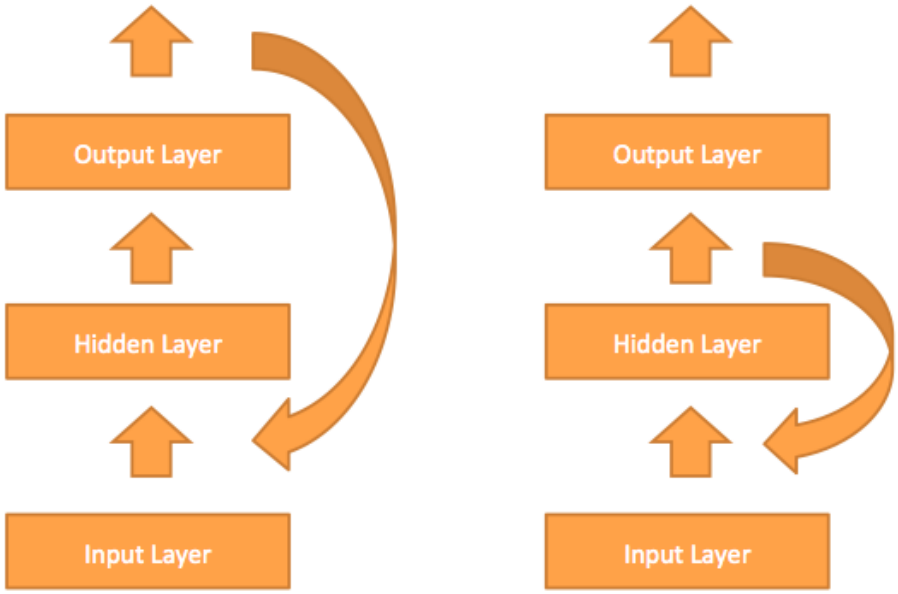
$$h^t = \sigma(W_h X + W_r y^{t-1})$$
$$y = \sigma(W_y h^t)$$

A few years later, another RNN was introduced by Elman (1990), when he formalized the recurrent structure slightly differently. Later, his network is known as Elman Network. Elman network is formalized as following:

$$h^t = \sigma(W_h X + W_r h^{t-1})$$
$$y = \sigma(W_y h^t)$$

The only difference is that whether the information of previous time step is provided by previous output or previous hidden layer. This difference is further illustrated in Figure 22. The difference is illustrated to respect the historical contribution of these works. One may notice that there is no fundamental difference between these two structures since $y_t = W_y h_t$, therefore, the only difference lies in the choice of W_r. (Originally, Elman only introduces his network with $W_r = \mathbf{I}$, but more general cases could be derived from there.)

Nevertheless, the step from Jordan Network to Elman Network is still remarkable as it introduces the possibility of passing information from hidden layers, which significantly improve the flexibility of structure design in later work.



(a) Structure of Jordan Network (b) Structure of Elman Network

Figure 22: The difference of recurrent structure from Jordan Network and Elman Network.

6.1.1 Backpropagation through Time

The recurrent structure makes traditional backpropagation infeasible because of that with the recurrent structure, there is not an end point where the backpropagation can stop.

Intuitively, one solution is to unfold the recurrent structure and expand it as a feedforward neural network with certain time steps and then apply traditional backpropagation onto this unfolded neural network. This solution is known as Backpropagation through Time (BPTT), independently invented by several researchers including (Robinson and Fallside, 1987; Werbos, 1988; Mozer, 1989)

However, as recurrent neural network usually has a more complex cost surface, naive backpropagation may not work well. Later in this paper, we will see that the recurrent structure introduces some critical problems, for example, the vanishing gradient problem, which makes optimization for RNN a great challenge in the society.

6.2 Bidirectional Recurrent Neural Network

If we unfold an RNN, then we can get the structure of a feedforward neural network with infinite depth. Therefore, we can build a conceptual connection between RNN and feedforward network with infinite layers. Then since through the neural network history, bidirectional neural networks have been playing important roles (like Hopfield Network, RBM, DBM), a follow-up question is that what recurrent structures that correspond to the infinite layer of bidirectional models are. The answer is Bidirectional Recurrent Neural Network.

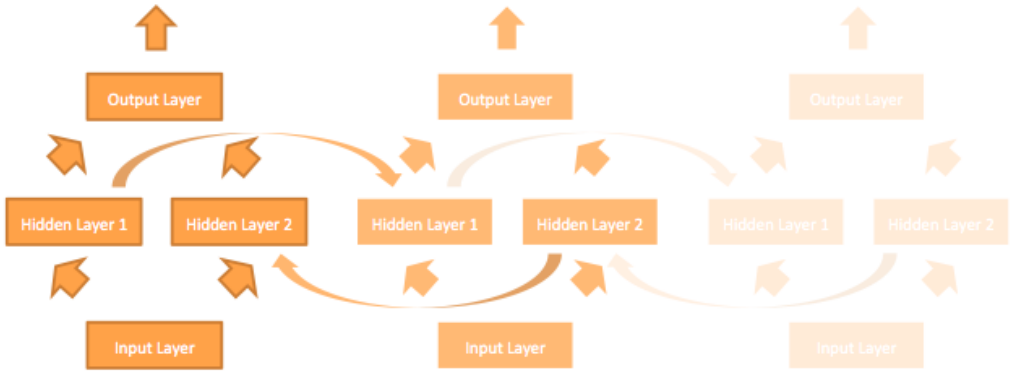


Figure 23: The unfolded structured of BRNN. The temporal order is from left to right. Hidden layer 1 is unfolded in the standard way of an RNN. Hidden layer 2 is unfolded to simulate the reverse connection.

Bidirectional Recurrent Neural Network (BRNN) was invented by Schuster and Paliwal (1997) with the goal to introduce a structure that was unfolded to be a bidirectional neural network. Therefore, when it is applied to time series data, not only the information can be passed following the natural temporal sequences, but the further information can also reversely provide knowledge to previous time steps.

Figure 23 shows the unfolded structure of a BRNN. Hidden layer 1 is unfolded in the standard way of an RNN. Hidden layer 2 is unfolded to simulate the reverse connection. Transparency (in Figure 23) is applied to emphasize that unfolding an RNN is only a concept that is used for illustration purpose. The actual model handles data from different time steps with the same single model.

BRNN is formulated as following:

$$
\begin{aligned}
h_1^t &= \sigma(W_{h1}X + W_{r1}h_1^{t-1}) \\
h_2^t &= \sigma(W_{h2}X + W_{r2}h_2^{t+1}) \\
y &= \sigma(W_{y1}h_1^t + W_{y2}h_2^t)
\end{aligned}
$$

where the subscript 1 and 2 denote the variables associated with hidden layer 1 and 2 respectively.

With the introduction of "recurrent" connections back from the future, Backpropagation through Time is no longer directly feasible. The solution is to treat this model as a combination of two RNNs: a standard one and a reverse one, then apply BPTT onto each of them. Weights are updated simultaneously once two gradients are computed.

6.3 Long Short-Term Memory

Another breakthrough in RNN family was introduced in the same year as BRNN. Hochreiter and Schmidhuber (1997) introduced a new neuron for RNN family, named Long Short-Term Memory (LSTM). When it was invented, the term "LSTM" is used to refer the algorithm

that is designed to overcome vanishing gradient problem, with the help of a special designed **memory cell**. Nowadays, "LSTM" is widely used to denote any recurrent network that with that memory cell, which is nowadays referred as an LSTM cell.

LSTM was introduced to overcome the problem that RNNs cannot long term dependencies (Bengio et al., 1994). To overcome this issue, it requires the specially designed memory cell, as illustrated in Figure 24 (a).

LSTM consists of several critical components.

- states: values that are used to offer the information for output.
  - ⋆ input data: it is denoted as x.
  - ⋆ hidden state: values of previous hidden layer. This is the same as traditional RNN. It is denoted as h.
  - ⋆ input state: values that are (linear) combination of hidden state and input of current time step. It is denoted as i, and we have:

$$i^t = \sigma(W_{ix}x^t + W_{ih}h^{t-1}) \tag{9}$$

  - ⋆ internal state: Values that serve as "memory". It is denoted as m
- gates: values that are used to decide the information flow of states.
  - ⋆ input gate: it decides whether input state enters internal state. It is denoted as g, and we have:

$$g^t = \sigma(W_{gi}i^t) \tag{10}$$

  - ⋆ forget gate: it decides whether internal state forgets the previous internal state. It is denoted as f, and we have:

$$f^t = \sigma(W_{fi}i^t) \tag{11}$$

  - ⋆ output gate: it decides whether internal state passes its value to output and hidden state of next time step. It is denoted as o and we have:

$$o^t = \sigma(W_{oi}i^t) \tag{12}$$

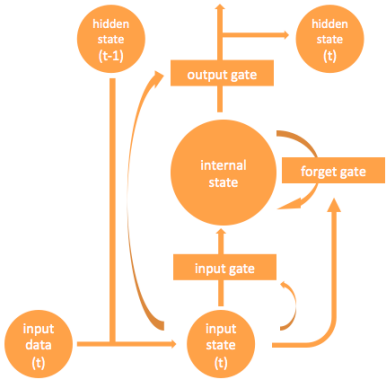
Finally, considering how gates decide the information flow of states, we have the last two equations to complete the formulation of LSTM:

$$m^t = g^t \odot i^t + f^t m^{t-1} \tag{13}$$

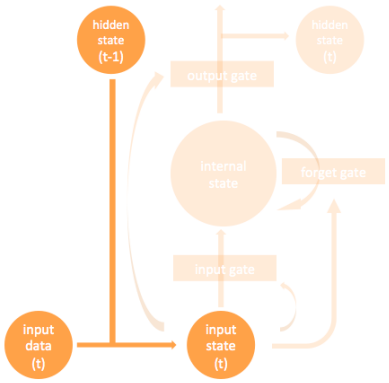
$$h^t = o^t \odot m^t \tag{14}$$

where $\odot$ denotes element-wise product.

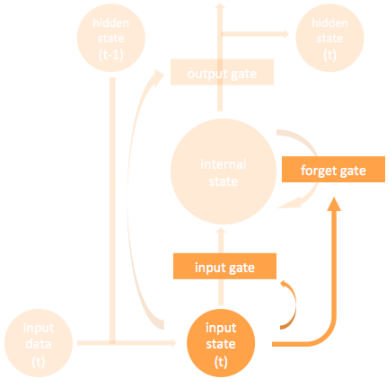
Figure 24 describes the details about how LSTM cell works. Figure 24 (b) shows that how the input state is constructed, as described in Equation 9. Figure 24 (c) shows how



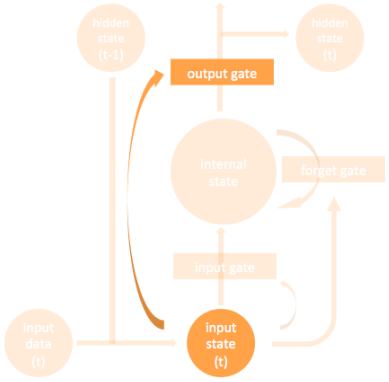
(a) LSTM "memory" cell



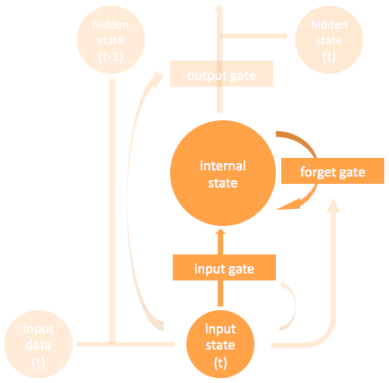
(b) Input data and previous hidden state form into input state



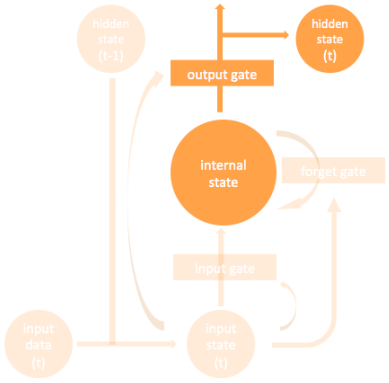
(c) Calculating input gate and forget gate



(d) Calculating output gate



(e) Update internal state



(f) Output and update hidden state

Figure 24: The LSTM cell and its detailed functions.

input gate and forget gate are computed, as described in Equation 10 and Equation 11. Figure 24 (d) shows how output gate is computed, as described in Equation 12. Figure 24 (e) shows how internal state is updated, as described in Equation 13. Figure 24 (f) shows how output and hidden state are updated, as described in Equation 14.

All the weights are parameters that need to be learned during training. Therefore, theoretically, LSTM can learn to memorize long time dependency if necessary and can learn to forget the past when necessary, making itself a powerful model.

With this important theoretical guarantee, many works have been attempted to improve LSTM. For example, Gers and Schmidhuber (2000) added a peephole connection that allows the gate to use information from the internal state. Cho et al. (2014) introduced the Gated Recurrent Unit, known as GRU, which simplified LSTM by merging internal state and hidden state into one state, and merging forget gate and input gate into a simple update gate. Integrating LSTM cell into bidirectional RNN is also an intuitive follow-up to look into (Graves et al., 2013).

Interestingly, despite the novel LSTM variants proposed now and then, Greff et al. (2015) conducted a large-scale experiment investigating the performance of LSTMs and got the conclusion that none of the variants can improve upon the standard LSTM architecture significantly. Probably, the improvement of LSTM is in another direction rather than updating the structure inside a cell. Attention models seem to be a direction to go.

6.4 Attention Models

Attention Models are loosely based on a bionic design to simulate the behavior of human vision attention mechanism: when humans look at an image, we do not scan it bit by bit or stare at the whole image, but we focus on some major part of it and gradually build the context after capturing the gist. Attention mechanisms were first discussed by Larochelle and Hinton (2010) and Denil et al. (2012). The attention models mostly refer to the models that were introduced in (Bahdanau et al., 2014) for machine translation and soon applied to many different domains like (Chorowski et al., 2015) for speech recognition and (Xu et al., 2015) for image caption generation.

Attention models are mostly used for sequence output prediction. Instead of seeing the whole sequential data and make one single prediction (for example, language model), the model needs to make a sequential prediction for the sequential input for tasks like machine translation or image caption generation. Therefore, the attention model is mostly used to answer the question on where to pay attention to based on previously predicted labels or hidden states.

The output sequence may not have to be linked one-to-one to the input sequence, and the input data may not even be a sequence. Therefore, usually, an encoder-decoder framework (Cho et al., 2015) is necessary. The encoder is used to encode the data into representations and decoder is used to make sequential predictions. Attention mechanism is used to locate a region of the representation for predicting the label in current time step.

Figure 25 shows a basic attention model under encoder-decoder network structure. The representation encoder encodes is all accessible to attention model, and attention model only selects some regions to pass onto the LSTM cell for further usage of prediction making.

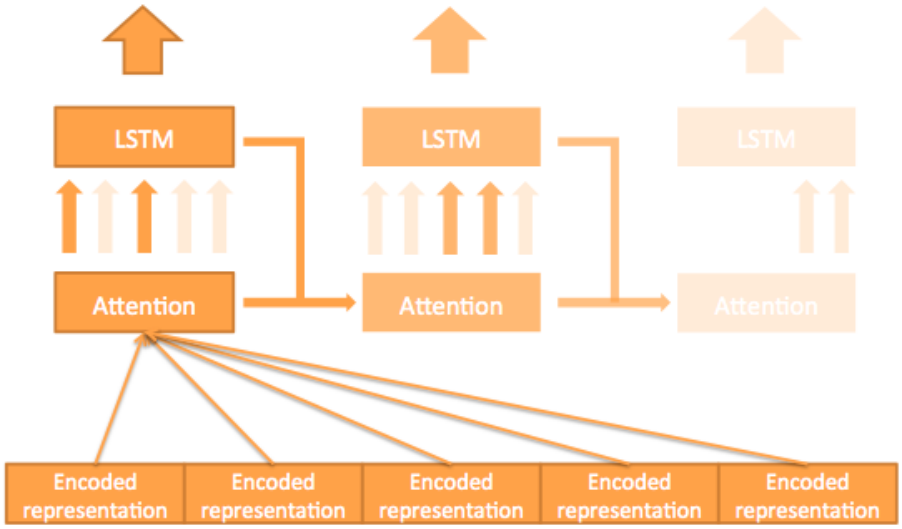


Figure 25: The unfolded structured of an attention model. Transparency is used to show that unfolding is only conceptual. The representation encoder learns are all available to the decoder across all time steps. Attention module only selects some to pass onto LSTM cell for prediction.

Therefore, all the magic of attention models is about how this attention module in Figure 25 helps to localize the informative representations.

To formalize how it works, we use r to denote the encoded representation (there is a total of M regions of representation), use h to denote hidden states of LSTM cell. Then, the attention module can generate the unscaled weights for ith region of the encoded representation as:

$$\beta_i^t = f(h^{t-1}, r, \{\alpha_j^{t-1}\}_{j=1}^M)$$

where α_j^{t-1} is the attention weights computed at the previous time step, and can be computed at current time step as a simple softmax function:

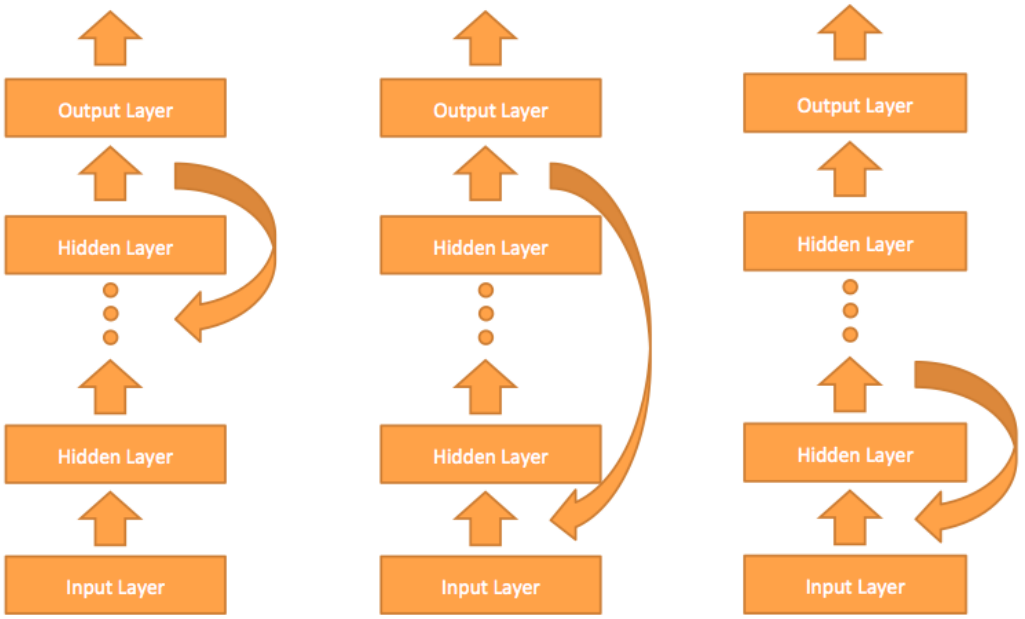
$$\alpha_i^t = \frac{\exp(\beta_i^t)}{\sum_j^M \exp(\beta_j^t)}$$

Therefore, we can further use the weights α to reweight the representation r for prediction. There are two ways for the representation to be reweighted:

- Soft attention: The result is a simple weighted sum of the context vectors such that:

$$r^t = \sum_j^M \alpha_j^t c_j$$

- Hard attention: The model is forced to make a hard decision by only localizing one region: sampling one region out following multinoulli distribution.



(a) Deep input architecture (b) Deep recurrent architecture (c) Deep output architecture

Figure 26: Three different formulations of deep recurrent neural network.

One problem about hard attention is that sampling out of multinoulli distribution is not differentiable. Therefore, the gradient based method can be hardly applied. Variational methods (Ba et al., 2014) or policy gradient based method (Sutton et al., 1999) can be considered.

6.5 Deep RNNs and the future of RNNs

In this very last section of the evolutionary path of RNN family, we will visit some ideas that have not been fully explored.

6.5.1 Deep Recurrent Neural Network

Although recurrent neural network suffers many issues that deep neural network has because of the recurrent connections, current RNNs are still not deep models regarding representation learning compared to models in other families.

Pascanu et al. (2013a) formalizes the idea of constructing deep RNNs by extending current RNNs. Figure 26 shows three different directions to construct a deep recurrent neural network by increasing the layers of the input component (Figure 26 (a)), recurrent component (Figure 26 (b)) and output component (Figure 26 (c)) respectively.

6.5.2 The Future of RNNs

RNNs have been improved in a variety of different ways, like assembling the pieces together with Conditional Random Field (Yang et al., 2016), and together with CNN components (Ma and Hovy, 2016). In addition, convolutional operation can be directly built into LSTM, resulting ConvLSTM (Xingjian et al., 2015), and then this ConvLSTM can be also connected with a variety of different components (De Brabandere et al., 2016; Kalchbrenner et al., 2016).

One of the most fundamental problems of training RNNs is the vanishing/exploding gradient problem, introduced in detail in (Bengio et al., 1994). The problem basically states that for traditional activation functions, the gradient is bounded. When gradients are computed by backpropagation following chain rule, the error signal decreases exponentially within the time steps the BPTT can trace back, so the long-term dependency is lost. LSTM and ReLU are known to be good solutions for vanishing/exploding gradient problem. However, these solutions introduce ways to bypass this problem with clever design, instead of solving it fundamentally. While these methods work well practically, the fundamental problem for a general RNN is still to be solved. Pascanu et al. (2013b) attempted some solutions, but there are still more to be done.

7. Generative Adversarial Networks and Modern Architectures

After the revisit of the evolutionary path of the classical models, we proceed to shift focus on more recent inventions of models, starting from the recent famous model: Generative Adversarial Network. Then, we will introduce several interesting architectures that may guide the direction of further research.

7.1 Generative Adversarial Networks

Generative Adversarial Network (GAN) was introduced by Goodfellow et al. (2014). It is a framework that is used to train generative models with the help of a discriminative classifier that is optimized to differentiate real images and adversarial images[10].

The main idea behind a GAN is to have two competing neural network models. One generates samples (therefore called generator). The other one distinguishes generated samples from real data by performing classification on the labels of image sources (therefore called discriminator). These two models are trained simultaneously, adversarially and force each other to improve by improving themselves. Ideally, the outcome of this adversarial training will result in a generator that can generate images that are indistinguishable from real images. An analogy of this adversarial training will be the development procedure of technologies of counterfeit currency and anti-counterfeit machines.

Figure 27 shows the basic structure of GAN. The components colored in cyan are the generative model that is the goal adversarial training. Components colored in purple are the components used to assist this training. The discriminator also needs to be trained, but training the classifier is not the major goal of this framework.

To formalize it, we use $G(\cdot, \theta)$ to denote the generator and use $D(\cdot, \delta)$ to denote discriminator, whose parameters are θ and δ respectively. We use x to denote an actual image, and use z to denote the latent variables that are fed into $G(\cdot, \theta)$. Therefore, an adversarial training is to optimize the following log-likelihood functions.

$$\min_{\theta} \max_{\delta} (\mathbb{E}_{x \sim p(x)}[\log D(x, \delta)] + \mathbb{E}_{z \sim q(z)}[\log D(G(z, \theta), \delta)])$$

where $p(x)$ is the data distribution and $q(z)$ is the distribution of latent variables.

This main cost function resembles the behavior of minimax theory, which is widely used in game theory (Maschler et al., 2013).

7.1.1 Parameter Learning

Although cost function of generator and discriminator has been formulated into one function, the parameters θ and δ are independent given the log-likelihood. Therefore, θ and δ can be updated alternatively. The convergence to a local optimum of alternative updating has been proved in the original paper (Goodfellow et al., 2014).

However, the learning process still has some problems (not surprising since this framework is still quite young). As Salimans et al. (2016) pointed out, training GANs requires finding a Nash equilibrium[11] of a non-convex game with continuous, high dimensional parameters. GANs are typically trained using gradient descent techniques that are designed

10. The idea of adversarial images has been mentioned in Figure 20

11. One of the most important concept in game theory and economics, see (Nash et al., 1950; Nash, 1951)

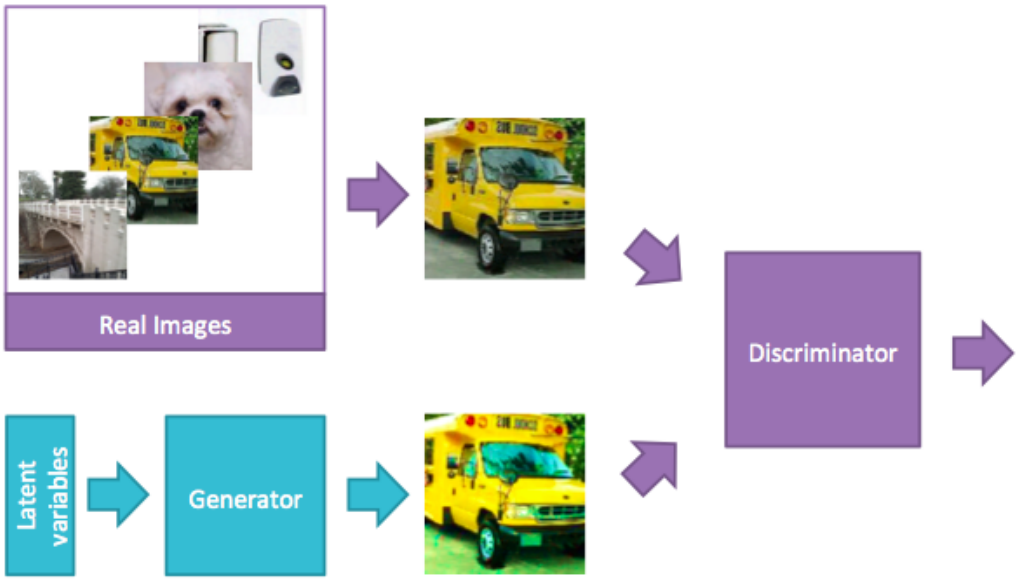


Figure 27: An illustration of the GAN architecture. The cyan components are the generative model part, which is the goal of adversarial training. The purple components are extra parts that are introduced by GAN architecture to assist adversarial training.

to find a low value of a cost function. When the task is to seek for a Nash equilibrium, these algorithms may fail to converge. To encourage the convergence, Salimans et al. (2016) proposed several techniques. One the the most influential ones is *minibatch discrimination*. Minibatch discrimination allows the discriminator to look at an entire batch of samples in order to decide whether they come from the generator or the real data. This simple technique turns out to play an important role in adversarial training. In addition, they also proposed several other techniques, for example: *Feature mapping* requires the generator to generate data to match the statistics of real data; *historical averaging*, inspired by old techniques in game theory (Brown, 1951), requires the model to consider historical parameters; *One-sided label smoothing* turns the discrete label into continuous ones.

7.1.2 Variants of GAN

GAN is still a very young framework. Therefore, there may not be many remarkable follow-ups to discuss. In addition, whether a model is distinguishable usually can only be answered by the test of time. In this paper, we selectively present a few models that have attracted wide attention.

InfoGan: Chen et al. (2016) introduced the InfoGan, which is an extension of GAN that learns disentangled and interpretable representations. The InfoGAN imposes additional structure on this latent variables by adding new objectives that involve maximizing the mutual information between small subsets of the representation variables and the obser-

vation, therefore, in contrast to traditional GAN that only generates a data point from latent variables without understanding what each values stand for, InfoGan can generate data with a specification of requirements. This goal has also been achieved by (Kulkarni et al., 2015), but InfoGao achieved this even without the knowledge of the existence of these requirements, which is remarkable.

Energy based GAN Zhao et al. (2016) proposed the Energy-based GAN (short as EBGAN) that formulates the original cost of GAN into a more machine-learning flavor context. However, as the discriminator is only presented as the reconstructed error from VAE, this direction is still worth exploration. Dai et al. (2017) shows a more general framework with explicit minimax formulation (in contrast to previous implicit formulation based on maximum likelihood estimation) about energy based GAN and avoids the degeneration problem of discriminator.

Wasserstein GAN Arjovsky et al. (2017) proposed to Wasserstein GAN. They first showed that the usual way of training a GAN is to force the generator to minimize the KL divergence between the generated data distribution and real data distribution. KL divergence, however, has a well-known disadvantage: for $KL(P||Q)$, the evaluated values is infinite for any point which has zero density in Q. They proposed to use Wasserstein distance to overcome the shortcoming of KL divergence.

Although GAN is still a relative young topic, there are a great amount of novel works generated rapidly nowadays. In addition to the topics that we find significant, one may refer to (Goodfellow, 2016) for a more thorough and focused discussion.

7.1.3 Game Theory and MiniMax

At the very last section about GAN framework, we briefly re-visit the field that GAN partially originates: the minimax of game theory. We did not introduce these before we introduce GAN is because that although GAN uses some established formalization of minimax, the idea of training a discriminator to improve a generator is unique.

Briefly, as John von Neumann showed in 1928 (Kjeldsen, 2001), if two players are competing in a game by taking actions, then the reward Player i can get (r_i) is:

$$r_i = \max_{a_i} \min_{a_j} v_i(a_i, a_j) = \min_{a_j} \max_{a_i} v_i(a_i, a_j)$$

This equation specifies the maximum value Player i can get given Player j's action. In other words, this is also smallest value that Player j can force Player i to get.

Despite the GAN's uniqueness of introducing discriminator and generator, this resemblance may encourage researchers to look back into game theory to search for inspirations that can significantly improve the training strategy of GAN[12].

7.2 Variational Autoencoder

Variational Autoencoder (VAE) is introduced by Kingma and Welling (2013), which also introduces a great possibility for generative models. More importantly, it bridges proba-

12. As a side note, we find that variational inference methods are very popular in recent works about GAN. One may want to thoroughly revisit these methods in classical textbooks, like (Wainwright et al., 2008) or (Murphy, 2012), for inspirations.

bilistic Bayesian inference and deep neural networks, which grants both societies a much wider field to explore.

7.2.1 Autoencoder

For the completeness of this paper, we first briefly discuss traditional Autoencoder, although VAE is mostly conceptually related to traditional Autoencoder. One can choose to skip these paragraphs for the focus of VAE.

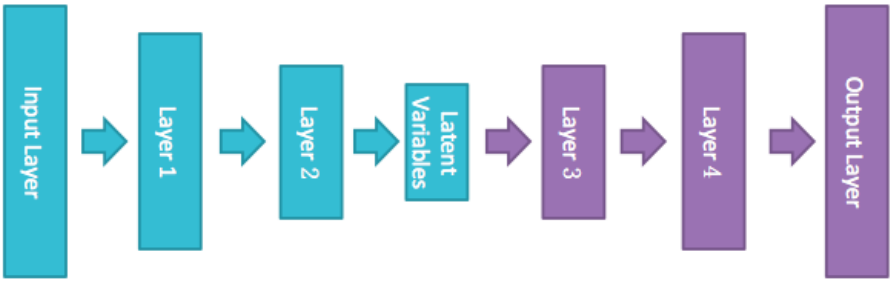
An autoencoder, firstly introduced in (Rumelhart et al., 1985), is a feedforward network that can learn a compressed, distributed representation of data, usually with the goal of dimensionality reduction or manifold learning. An autoencoder usually has one hidden layer between input and output layer. Hidden layer usually has a more compact representation than input and output layers, i.e. hidden layer has fewer units than input or output layer. Input and output layer usually have the same setting, which allows an autoencoder to be trained unsupervisedly with same data fed in at the input and compared with what is at the output layer. The training process is the same as traditional neural network with backpropagation; the only difference lies in the error is computed by comparing the output to the data itself. Mitchell et al. (1997) showed a nice illustration of autoencoder. He built a three-layer structure (eight unit for input and output layer, and three unit for the hidden layer in between), then he fed the one-hot vector representation into the input and output layer, the hidden layer turned out to approximate the data with inputs' binary representation.

A **stacked autoencoder** is the deep counterpart of autoencoder, it can be built simply by stacking up layers. For every layer, its input is the learned representation of former layer and it learns a more compact representation of the existing learned representation. Thus a stacked autoencoders will have a pyramidal architecture. Figure 28 shows the architecture of a stacked autoencoder. The cyan components are the encoder part that can be used to learn the representation of data. The purple components are the decoder that are used to assist the training. A **stacked sparse autoencoder**, discussed by (Gravelines, 1991) is a stacked autoencoder where sparsity regularizations are introduced into the autoencoders to learn a sparse representation. A **stacked denoising autoencoder**, introduced by (Vincent et al., 2010) is an autoencoder where the data at input layer is replaced with noised data while the data at output layer stays the same, therefore the autoencoder can be trained with much more generalization power.

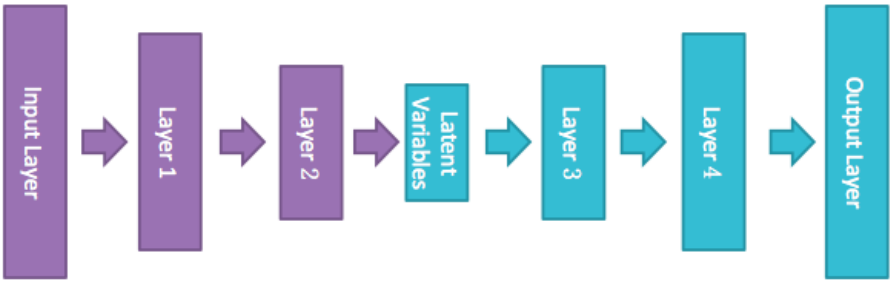
7.2.2 VAE

VAE, firstly introduced by Kingma and Welling (2013), inherits the name "autoencoder" because it also follows the encoder-decoder framework as illustrated in Figure 28. In addition to the manifold learning task that has been widely explored with traditional autoencoders (cyan components in Figure 28 (a)), VAE can also be used as a generative model to conjure up brand new examples, consistent with the observed training set, that do not exist in nature (cyan components in Figure 28 (b)).

To understand VAE, we first split it into two parts: the **encoder** (cyan components in Figure 28 (a)) and the **decoder** (purple components in Figure 28 (a)). For VAE, encoder and decoder can be built with different architectures distinctly different from classical



(a) Conceptual structure for autoencoder, VAE as a manifolding learning model



(b) Conceptual structure for VAE as a generative model

Figure 28: An illustration of the concept of autoencoder. The cyan components are actual model. The purple components are used to assist the training.

autoencoder. The encoder and decoder in VAE are probabilistic functions, instead of deterministic ones like neural networks. This distinction makes VAE to be more similar to probabilistic graphical models where the encoder is given by $q(z|x)$ and decoder is given by $p(x|z)$ (where x denotes data, z denotes latent variables, $p(\cdot)$ and $q(\cdot)$ are distributions.). However, the remarkable notion is that VAE achieves these probabilistic functions with neural networks, which serves like a bridge between Bayesian world and neural network world.

Despite the breakthrough, the underlying techniques used are not magical. To simulate $q(z|x)$, one assumption is introduced that states z is following a Normal distribution. Therefore,

$$q(z|x) \sim N(f(x;\theta), g(x;\delta))$$

where $f(\cdot;\theta)$ and $g(\cdot;\delta)$ are neural networks with parameters θ and δ respectively. As a result, although $f(\cdot;\theta)$ and $g(\cdot;\delta)$ are deterministic functions, the Gaussian distribution naturally introduces probabilistic mechanism. To make this Gaussian distribution a Normal one, the optimization needs to minimize the KL-divergence between this distribution and a Normal distribution.

More importantly, $f(\cdot;\theta)$ and $g(\cdot;\delta)$ can be built with any architecture or any number of layers and any forms of non-linearity given by choices of activation functions, the modeling power of traditional graphic models is dramatically improved.

For a generative model, the probabilistic mechanism is introduced by first sampling z from the Normal distribution, and then $\tilde{x} \sim h(z;\phi)$, where $\tilde{x}$ is the generated data, $h(\cdot;\phi)$ is another neural network with parameter ϕ.

Therefore, the final cost function we optimize is the likelihood of $\tilde{x}$ plus the KL divergence between $N(f(x;\theta), g(x;\delta))$ and a Normal distribution (Doersch, 2016). Gradient descent and its variants can be directly applied.

7.2.3 Variants of VAE

To conclude VAE, we proceed to look into some important variants of VAE.

Conditional VAE Conditional VAE is an extension of VAE that allows conditional encoder and decoder, introduced by (Kingma et al., 2014; Sohn et al., 2015). It replaces the encoder $q(z|x)$ with $q(z|x,y)$ and replaces the decoder $p(x|z)$ with $p(x|z,y)$. Interestingly, y appears both in training and generating phases as extra information.

Importance Weighted VAE Importance Weighted VAE, proposed by Burda et al. (2015), improves VAE with a strictly tighter log-likelihood lower bound derived from importance weighting (related to (Bornschein and Bengio, 2014)) by introducing a weight vector governing the importance of samples, as a deterministic function of x and z.

7.3 Select-Additive Networks

Select-Additive Network is introduced by Wang et al. (2016). It is a network that is used to improve the generalization power of an existing discriminative network, especially when training data are from different distributions or the training data are insufficient for a neural network to get trained. Select-Additive Network is inspired by traditional Linear Mixed Model, which is used to learn a generalized linear regression across different distributions of data.

7.3.1 Linear Mixed Model

Linear Mixed Model was introduced a long time ago (Cnaan et al., 1997; Booth and Hobert, 1999), when it was mainly applied to study the biological behaviors of different populations. Since different populations will have different behaviors and traditional linear regression cannot be trained with generalization across different distributions back then with a limit amount of data, linear mixed was introduced.

In contrast to traditional linear regression which models data as $y = X\beta + \epsilon$, Linear Mixed Model models the data considering the variance introduced by different distributions, as

$$y \sim N(X\beta, ZZ^T\sigma_\mu + \sigma_\epsilon) \tag{15}$$

where Z is the indicator of where the distribution comes from, σ_μ specifies the variance introduced by distributions.

This formalization may seem awkward in the first glance compared to a more plausibly reasonable formalization like $y = [X, Z]^T[\beta, \mu]$. However, this formalization turns out to have a better generalization power especially when there are confounding factors. Linear Mixed Model can effectively correct the confounding factors and build a more generalized model to be applied to data across distributions. In other words, it potentially discovers the causal relationship (Duckworth et al., 2010). It has been widely applied to areas where the causal relationship is critical (Lippert et al., 2011; Zhou and Stephens, 2012).

With this critical usage, many more generalized version of Linear Mixed Models have been introduced in the linear cases, like (McCulloch and Neuhaus, 2001; Nguyen et al., 2008; Wang and Yang, 2016)

7.3.2 Select-Additive Networks

Select-Additive Network is the neural counterpart of Linear Mixed Model. It utilizes the idea of VAE as a bridge to introduce a more powerful model into Linear Mixed Model family. On the other hand, it also introduces a new architecture that can help deep learning models to generalize well with limited data from distributions and confounding factors within the data. A deep learning model can hardly be trained to discard confounding factors by initial training procedure because confounding factors, by definition, are associated with labels of the data, but in a non-causal manner. Traditional training of deep learning cannot distinguish features by whether a causal manner exists, therefore cannot remove the confounding effects. The introduction of Select-Additive Network brings in this possibility. If VAE connects probabilistic graphical models and neural networks, Select-Additive Network may connect neural network with causality oriented research.

Figure 29 is an illustration of the select-additive network. The cyan components are the actual model that needs a better generalization across distributions, the purple components are used to assist the training.

The idea is that the purple part plays the role as the variance part of the Gaussian distribution in Equation 15 plays. It forces the model to discard the information that is related to confounding factors. It first selects the representation learned by original model from confounding factors by minimizing the differences between two representations. Then it forces the original model to discard these representation by adding Gaussian noises onto these representations.

To formalize this model, we first split the original model into two parts: the encoder and the discriminator. The encoder is the bottom part from the input layer up to where the purple part connects to the original model (layer 2, as showed in Figure 29), denoted as $g(\cdot;\theta)$. Then the discriminator is the model that continues from where the purple part connects up to the output, denoted as $f(\cdot;\phi)$. The purple part is also a neural network, denoted as $h(\cdot;\delta)$. θ, ϕ and δ are parameters. We use X to denote input, use Z to denote the distributions where these data come from and use y to denote the labels of these data.

Then, following Equation 15, we have the cost function as:

$$\underset{\phi,\theta,\delta}{\operatorname{argmin}} \frac{1}{2}(y - f(g(X,\theta) + h(Z,\delta)\mu, \phi))^2$$

where $\mu \sim N(0, 1)$

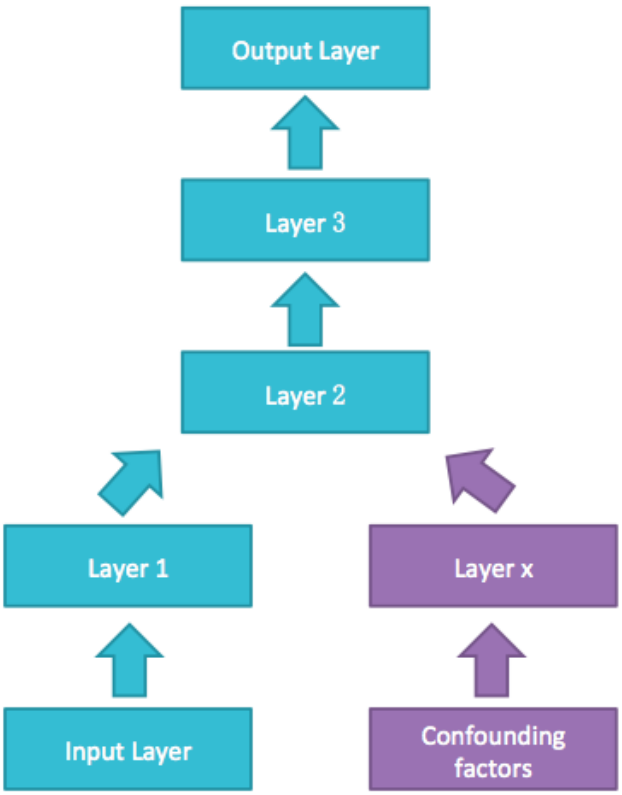


Figure 29: An illustration of the Select-Additive Network architecture. The cyan components are the actual model that needs a better generalization across distributions, the purple components are used to assist the training.

However, directly optimize this model will be hard. Therefore Wang et al. (2016) introduces a select-additive algorithm that splits the training procedures into phases and updates the parameters alternatively. The algorithm first updates with:

$$\underset{\phi,\theta}{\operatorname{argmin}} \frac{1}{2}(y - f(g(X;\theta);\phi))^2$$

Then the algorithm performs

$$\arg\min_{\delta} \frac{1}{2}(g(X;\theta) - h(Z;\delta))^2 + \lambda||\delta||_1$$

for the purple part to select which parts of the representation are associated with confounding factors. λ is a scalar that controls the weight of the sparsity regularizer. Finally, the algorithm performs

$$\underset{\phi}{\operatorname{argmin}} \frac{1}{2}(y - f(g(X;\theta) + h(Z;\delta) \odot \epsilon;\phi))^2$$

to tune the parameters ϕ to discard the representation that is associated with confounding factors. $\odot$ denotes element-wise product.

Therefore, this algorithm grants the possibility for Select-Additive Network to correct confounding factors and further to bridge the neural network research with causality-oriented researches.

The field of Select-Additive Network is still very young. Therefore, there are not follow-ups to be discussed in this paper. In other words, there are a lot of further directions could be done to extend this model. For example, although the phased learning algorithm experimentally works better than naively optimizing the unified cost function, a unified algorithm is still necessary to reduce the choices of hyperparameters. Another example will be applying some clustering-type preprocessing to identity Z automatically, and further integrating this preprocessing step into the neural architecture itself aiming for a unified process.

Most importantly, a recent work that claims "understanding deep learning required rethinking generalization" (Zhang et al., 2016a) raised some interesting arguments about deep learning models by showing that deep learning models seem to memorize too much information to reach a good performance while maintaining the good generalization power. This contradictory argument (memorizing information v.s. generalizing information) might be a result of how the generalization is defined. On homogeneous data, memorizing information surely helps generalization, while the real challenge of deep learning, or machine learning in general, should be generalization on heterogeneous data. Select-Additive Network, naturally born as a model to work with heterogeneous data, is expected to meet a higher criterion than current generalization task.

7.4 Other Novel Architectures

There is a recent trend of introducing knowledge into neural networks to assist its learning procedure. Several architectures are introduced and showed vast improvement upon previous solutions.

Hu et al. (2016) introduced a teacher-student framework where they proposed an iterative distillation method that transfers the structured information of logic rules into the weights of neural networks and enables neural network to learn from both labeled data and from prior knowledge that is encoded as logic rules. They improved the performance with this framework both for CNN with sentiment analysis and RNN for NER task.

Shazeer et al. (2017) introduced a Sparsely-Gated Mixture-of-Experts layer that can be built into other general deep learning architecture. They showed the good performance with plugging it into recurrent architectures applied language tasks. While their contributions in both modeling and optimization are exciting, the mainly idea behind is Sparsely-Gated Mixture-of-Experts layer is more or less the neural network version of standard Mixture-of-Expert (MoE) model that has been discussed in classical probabilistic graphical model (PGM) textbooks for years. While this work opens a whole area of integrating neural MoE into existing architecture for followers-up to explore, readers can also refer back to old machine learning techniques to seek for inspirations and attempt to open an era of PGM marrying deep learning.

8. Optimization of Neural Networks

The primary focus of this paper is deep learning models. However, optimization is an inevitable topic in the development history of deep learning models. In this section, we will briefly revisit the major topics of optimization of neural networks. During our introduction of the models, some algorithms have been discussed along with the models. Here, we will only discuss the remaining methods that have not been mentioned previously.

8.1 Gradient Methods

Despite the fact that neural networks have been developed for over fifty years, the optimization of neural networks still heavily rely on gradient descent methods within the algorithm of backpropagation. This paper does not intend to introduce the classical backpropagation, gradient descent method and its stochastic version and batch version and simple techniques like momentum method, but starts right after these topics.

Therefore, the discussion of following gradient methods starts from the vanilla gradient descent as following:

$$\theta^{t+1} = \theta^t - \eta \bigtriangledown_\theta^t$$

where $\bigtriangledown_\theta$ is the gradient of the parameter θ, η is a hyperparameter, usually known as learning rate.

8.1.1 Rprop

Rprop was introduced by Riedmiller and Braun (1993). It is a unique method even studied back today as it does not fully utilize the information of gradient, but only considers the sign of it. In other words, it updates the parameters following:

$$\theta^{t+1} = \theta^t - \eta I(\bigtriangledown_\theta^t > 0) + \eta I(\bigtriangledown_\theta^t < 0)$$

where $I(\cdot)$ stands for an indicator function.

This unique formalization allows the gradient method to overcome some cost curvatures that may not be easily solved with today's dominant methods. This two-decade-old method may be worth some further study these days.

8.1.2 AdaGrad

AdaGrad was introduced by Duchi et al. (2011). It follows the idea of introducing an adaptive learning rate mechanism that assigns higher learning rate to the parameters that have been updated more mildly and assigns lower learning rate to the parameters that have been updated dramatically. The measure of the degree of the update applied is the ℓ_2 norm of historical gradients, $S^t = ||\bigtriangledown_\theta^1, \bigtriangledown_\theta^2, ... \bigtriangledown_\theta^t||_2^2$, therefore we have the update rule as following:

$$\theta^{t+1} = \theta^t - \frac{\eta}{S^t + \epsilon} \bigtriangledown_\theta^t$$

where ϵ is small term to avoid η divided by zero.

AdaGrad has been showed with great improvement of robustness upon traditional gradient method (Dean et al., 2012). However, the problem is that as ℓ_2 norm accumulates, the fraction of η over ℓ_2 norm decays to a substantial small term.

8.1.3 AdaDelta

AdaDelta is an extension of AdaGrad that aims to reducing the decaying rate of learning rate, proposed in (Zeiler, 2012). Instead of accumulating the gradients of each time step as in AdaGrad, AdaDelta re-weights previously accumulation before adding current term onto previously accumulated result, resulting in:

$$(S^t)^2 = \beta(S^{t-1})^2 + (1-\beta)(\bigtriangledown_\theta^t)^2$$

where β is the weight for re-weighting. Then the update rule is the same as AdaGrad:

$$\theta^{t+1} = \theta^t - \frac{\eta}{S^t + \epsilon}\bigtriangledown_\theta^t$$

which is almost the same as another famous gradient variant named RMSprop[13].

8.1.4 Adam

Adam stands for Adaptive Moment Estimation, proposed in (Kingma and Ba, 2014). Adam is like a combination momentum method and AdaGrad method, but each component are re-weighted at time step t. Formally, at time step t, we have:

$$\begin{aligned}
\Delta_\theta^t =& \alpha\Delta_\theta^{t-1} + (1-\alpha)\bigtriangledown_\theta^t \\
(S^t)^2 =& \beta(S^{t-1})^2 + (1-\beta)(\bigtriangledown_\theta^t)^2 \\
\theta^{t+1} =& \theta^t - \frac{\eta}{S^t + \epsilon}\Delta_\theta^t
\end{aligned}$$

All these modern gradient variants have been published with a promising claim that is helpful to improve the convergence rate of previous methods. Empirically, these methods seem to be indeed helpful, however, in many cases, a good choice of these methods seems only to benefit to a limited extent.

8.2 Dropout

Dropout was introduced in (Hinton et al., 2012; Srivastava et al., 2014). The technique soon got influential, not only because of its good performance but also because of its simplicity of implementation. The idea is very simple: randomly dropping out some of the units while training. More formally: on each training case, each hidden unit is randomly omitted from the network with a probability of p.

As suggested by Hinton et al. (2012), Dropout can be seen as an efficient way to perform model averaging across a large number of different neural networks, where overfitting can be avoided with much less cost of computation.

13. It seems this method never gets published, the resources all trace back to Hinton's slides at http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf

Because of the actual performance it introduces, Dropout soon became very popular upon its introduction, a lot of work has attempted to understand its mechanism in different perspectives, including (Baldi and Sadowski, 2013; Cho, 2013; Ma et al., 2016). It has also been applied to train other models, like SVM (Chen et al., 2014).

8.3 Batch Normalization and Layer Normalization

Batch Normalization, introduced by Ioffe and Szegedy (2015), is another breakthrough of optimization of deep neural networks. They addressed the problem they named as *internal covariate shift*. Intuitively, the problem can be understood as the following two steps: 1) a learned function is barely useful if its input changes (In statistics, the input of a function is sometimes denoted as covariates). 2) each layer is a function and the changes of parameters of below layers change the input of current layer. This change could be dramatic as it may shift the distribution of inputs.

Ioffe and Szegedy (2015) proposed the Batch Normalization to solve this issue, formally following the steps:

$$\begin{aligned}
\mu_B =& \frac{1}{n}\sum_{i=1}^{n} x_i \\
\sigma_B^2 =& \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu_B)^2 \\
\hat{x_i} =& \frac{x_i - \mu_B}{\sigma_B + \epsilon} \\
y_i =& \sigma_L \hat{x_i} + \mu_L
\end{aligned}$$

where μ_B and σ_B denote the mean and variance of that batch. μ_L and σ_L two parameters learned by the algorithm to rescale and shift the output. x_i and y_i are inputs and outputs of that function respectively.

These steps are performed for every batch during training. Batch Normalization turned out to work very well in training empirically and soon became popularly.

As a follow-up, Ba et al. (2016) proposes the technique Layer Normalization, where they "transpose" batch normalization into layer normalization by computing the mean and variance used for normalization from all of the summed inputs to the neurons in a layer on a single training case. Therefore, this technique has a nature advantage of being applicable to recurrent neural network straightforwardly. However, it seems that this "transposed batch normalization" cannot be implemented as simple as Batch Normalization. Therefore, it has not become as influential as Batch Normalization is.

8.4 Optimization for "Optimal" Model Architecture

In the very last section of optimization techniques for neural networks, we revisit some old methods that have been attempted with the aim to learn the "optimal" model architecture. Many of these methods are known as constructive network approaches. Most of these methods have been proposed decades ago and did not raise enough impact back then. Nowadays, with more powerful computation resources, people start to consider these methods again.

Two remarks need to be made before we proceed: 1) Obviously, most of these methods can trace back to counterparts in non-parametric machine learning field, but because most of these methods did not perform enough to raise an impact, focusing a discussion on the evolutionary path may mislead readers. Instead, we will only list these methods for the readers who seek for inspiration. 2) Many of these methods are not exclusively optimization techniques because these methods are usually proposed with a particularly designed architecture. Technically speaking, these methods should be distributed to previous sections according to the models associated. However, because these methods can barely inspire modern modeling research, but may have a chance to inspire modern optimization research, we list these methods in this section.

8.4.1 Cascade-Correlation Learning

One of the earliest and most important works on this topic was proposed by Fahlman and Lebiere (1989). They introduced a model, as well as its corresponding algorithm named Cascade-Correlation Learning. The idea is that the algorithm starts with a minimum network and builds up towards a bigger network. Whenever another hidden unit is added, the parameters of previous hidden units are fixed, and the algorithm only searches for an optimal parameter for the newly-added hidden unit.

Interestingly, the unique architecture of Cascade-Correlation Learning grants the network to grow deeper and wider at the same time because every newly added hidden unit takes the data together with outputs of previously added units as input.

Two important questions of this algorithm are 1) when to fix the parameters of current hidden units and proceed to add and tune a newly added one 2) when to terminate the entire algorithm. These two questions are answered in a similar manner: the algorithm adds a new hidden unit when there are no significant changes in existing architecture and terminates when the overall performance is satisfying. This training process may introduce problems of overfitting, which might account for the fact that this method is seen much in modern deep learning research.

8.4.2 Tiling Algorithm

Mézard and Nadal (1989) presented the idea of Tiling Algorithm, which learns the parameters, the number of layers, as well as the number of hidden units in each layer simultaneously for feedforward neural network on Boolean functions. Later this algorithm was extended to multiple class version by Parekh et al. (1997).

The algorithm works in such a way that on every layer, it tries to build a layer of hidden units that can cluster the data into different clusters where there is only one label in one cluster. The algorithm keeps increasing the number of hidden units until such a clustering pattern can be achieved and proceed to add another layer.

Mézard and Nadal (1989) also offered a proof of theoretical guarantees for Tiling Algorithm. Basically, the theorem says that Tiling Algorithm can greedily improve the performance of a neural network.

8.4.3 Upstart Algorithm

Frean (1990) proposed the Upstart Algorithm. Long story short, this algorithm is simply a neural network version of the standard decision tree (Safavian and Landgrebe, 1990) where each tree node is replaced with a linear perceptron. Therefore, the tree is seen as a neural network because it uses the core component of neural networks as a tree node. As a result, standard way of building a tree is advertised as building a neural network automatically.

Similarly, Bengio et al. (2005) proposed a boosting algorithm where they replace the weak classifier as neurons.

8.4.4 Evolutionary Algorithm

Evolutionary Algorithm is a family of algorithms uses mechanisms inspired by biological evolution to search in a parameter space for the optimal solution. Some prominent examples in this family are genetic algorithm (Mitchell, 1998), which simulates natural selection and ant colony optimization algorithm (Colorni et al., 1991), which simulates the cooperation of an ant colony to explore surroundings.

Yao (1999) offered an extensive survey of the usage of evolution algorithm upon the optimization of neural networks, in which Yao introduced several encoding schemes that can enable the neural network architecture to be learned with evolutionary algorithms. The encoding schemes basically transfer the network architecture into vectors, so that a standard algorithm can take it as input and optimize it.

So far, we discussed some representative algorithms that are aimed to learn the network architecture automatically. Most of these algorithms eventually fade out of modern deep learning research, we conjecture two main reasons for this outcome: 1) Most of these algorithms tend to overfit the data. 2) Most of these algorithms are following a greedy search paradigm, which will be unlikely to find the optimal architecture.

However, with the rapid development of machine learning methods and computation resources in the last decade, we hope these constructive network methods we listed here can still inspire the readers for substantial contributions to modern deep learning research.

9. Conclusion

In this paper, we have revisited the evolutionary path of the nowadays deep learning models. We first revisited the paths for three major families of deep learning models: the deep generative model family, convolutional neural network family, and recurrent neural network family. Then we proceeded to introduce some newly developed architectures, featuring Generative Adversarial Nets. Finally, we revisited the some topics for optimization techniques.

This paper could serve two goals: 1) First, it documents the major milestones in the science history that have impacted the current development of deep learning. These milestones are not limited to the development in computer science fields. 2) More importantly, by revisiting the evolutionary path of the major milestone, this paper should be able to suggest the readers that how these remarkable works are developed among thousands of other contemporaneous publications. Here we briefly summarize three directions that many of these milestones pursue:

- **Occam's razor:** While it seems that part of the society tends to favor more complex models by layering up one architecture onto another and hoping backpropagation can find the optimal parameters, history says that masterminds tend to think simple: Dropout is widely recognized not only because of its performance, but more because of its simplicity in implementation and intuitive (tentative) reasoning. From Hopfield Network to Restricted Boltzmann Machine, models are simplified along the iterations until when RBM is ready to be piled-up.

- **Be ambitious:** If a model is proposed with substantially more parameters than contemporaneous ones, it must solve a problem that no others can solve nicely to be remarkable. LSTM is much more complex than traditional RNN, but it bypasses the vanishing gradient problem nicely. Deep Belief Network is famous not due to the fact the they are the first one to come up with the idea of putting one RBM onto another, but due to that they come up an algorithm that allow deep architectures to be trained effectively.

- **Widely read:** Many models are inspired by domain knowledge outside of machine learning or statistics field. Human visual cortex has greatly inspired the development of convolutional neural networks. Even the recent popular Residual Networks can find corresponding mechanism in human visual cortex. Generative Adversarial Network can also find some connection with game theory, which was developed fifty years ago.

We hope these directions can help some readers to impact more on current society. More directions should also be able to be summarized through our revisit of these milestones by readers.

Acknowledgements

Thanks to the demo from http://beej.us/blog/data/convolution-image-processing/ for a quick generation of examples in Figure 16. Thanks to Bojian Han at Carnegie Mellon University for the examples in Figure 21. Thanks to the blog at http://sebastianruder.com/optimizing-gradient-descent/index.html for a summary of gradient methods in Section 8.1. Thanks to

Yutong Zheng and Xupeng Tong at Carnegie Mellon University for suggesting some relevant contents.

References

Emile Aarts and Jan Korst. Simulated annealing and boltzmann machines. 1988.

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

James A Anderson and Edward Rosenfeld. *Talking nets: An oral history of neural networks*. MiT Press, 2000.

Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein gan. *arXiv preprint arXiv:1701.07875*, 2017.

Jimmy Ba, Volodymyr Mnih, and Koray Kavukcuoglu. Multiple object recognition with visual attention. *arXiv preprint arXiv:1412.7755*, 2014.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

Alexander Bain. *Mind and Body the Theories of Their Relation by Alexander Bain*. Henry S. King & Company, 1873.

Pierre Baldi and Peter J Sadowski. Understanding dropout. In *Advances in Neural Information Processing Systems*, pages 2814–2822, 2013.

Peter L Bartlett and Wolfgang Maass. Vapnik chervonenkis dimension of neural nets. *The handbook of brain theory and neural networks*, pages 1188–1192, 2003.

Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf: Speeded up robust features. In *European conference on computer vision*, pages 404–417. Springer, 2006.

Yoshua Bengio. Learning deep architectures for ai. *Foundations and trends® in Machine Learning*, 2(1):1–127, 2009.

Yoshua Bengio and Olivier Delalleau. On the expressive power of deep architectures. In *International Conference on Algorithmic Learning Theory*, pages 18–36. Springer, 2011.

Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

Yoshua Bengio, Nicolas L Roux, Pascal Vincent, Olivier Delalleau, and Patrice Marcotte. Convex neural networks. In *Advances in neural information processing systems*, pages 123–130, 2005.

Yoshua Bengio, Pascal Lamblin, Dan Popovici, Hugo Larochelle, et al. Greedy layer-wise training of deep networks. *Advances in neural information processing systems*, 19:153, 2007.

Monica Bianchini and Franco Scarselli. On the complexity of shallow and deep neural network classifiers. In *ESANN*, 2014.

CG Boeree. Psychology: the beginnings. *Retrieved April*, 26:2008, 2000.

James G Booth and James P Hobert. Maximizing generalized linear mixed model likelihoods with an automated monte carlo em algorithm. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 61(1):265–285, 1999.

Jörg Bornschein and Yoshua Bengio. Reweighted wake-sleep. *arXiv preprint arXiv:1406.2751*, 2014.

Nirmal K Bose et al. *Neural network fundamentals with graphs, algorithms, and applications.* Number 612.82 BOS. 1996.

Martin L Brady, Raghu Raghavan, and Joseph Slawny. Back propagation fails to separate where perceptrons succeed. *IEEE Transactions on Circuits and Systems*, 36(5):665–674, 1989.

George W Brown. Iterative solution of games by fictitious play. *Activity analysis of production and allocation*, 13(1):374–376, 1951.

John A Bullinaria. Self organizing maps: Fundamentals. *Introduction to Neural*, 2004.

Yuri Burda, Roger Grosse, and Ruslan Salakhutdinov. Importance weighted autoencoders. *arXiv preprint arXiv:1509.00519*, 2015.

WH Burnham. Memory, historically and experimentally considered. i. an historical sketch of the older conceptions of memory. *The American Journal of Psychology*, 2(1):39–90, 1888.

Alfredo Canziani, Adam Paszke, and Eugenio Culurciello. An analysis of deep neural network models for practical applications. *arXiv preprint arXiv:1605.07678*, 2016.

Miguel A Carreira-Perpinan and Geoffrey Hinton. On contrastive divergence learning. In *AISTATS*, volume 10, pages 33–40. Citeseer, 2005.

Juan Luis Castro, Carlos Javier Mantas, and JM Benıtez. Neural networks with a continuous squashing function in the output are universal approximators. *Neural Networks*, 13(6): 561–563, 2000.

Ning Chen, Jun Zhu, Jianfei Chen, and Bo Zhang. Dropout training for support vector machines. *arXiv preprint arXiv:1404.4171*, 2014.

Xi Chen, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, and Pieter Abbeel. Infogan: Interpretable representation learning by information maximizing generative adversarial nets. In *Advances In Neural Information Processing Systems*, pages 2172–2180, 2016.

KyungHyun Cho. Understanding dropout: training multi-layer perceptrons with auxiliary independent stochastic neurons. In *International Conference on Neural Information Processing*, pages 474–481. Springer, 2013.

Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

Kyunghyun Cho, Aaron Courville, and Yoshua Bengio. Describing multimedia content using attention-based encoder-decoder networks. *IEEE Transactions on Multimedia*, 17 (11):1875–1886, 2015.

Anna Choromanska, Mikael Henaff, Michael Mathieu, Gérard Ben Arous, and Yann LeCun. The loss surfaces of multilayer networks. In *AISTATS*, 2015.

Jan K Chorowski, Dzmitry Bahdanau, Dmitriy Serdyuk, Kyunghyun Cho, and Yoshua Bengio. Attention-based models for speech recognition. In *Advances in Neural Information Processing Systems*, pages 577–585, 2015.

Avital Cnaan, NM Laird, and Peter Slasor. Tutorial in biostatistics: Using the general linear mixed model to analyse unbalanced repeated measures and longitudinal data. *Stat Med*, 16:2349–2380, 1997.

Alberto Colorni, Marco Dorigo, Vittorio Maniezzo, et al. Distributed optimization by ant colonies. In *Proceedings of the first European conference on artificial life*, volume 142, pages 134–142. Paris, France, 1991.

David Daniel Cox and Thomas Dean. Neural networks and neuroscience-inspired computer vision. *Current Biology*, 24(18):R921–R929, 2014.

G Cybenko. *Continuous valued neural networks with two hidden layers are sufficient*. 1988.

George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

Zihang Dai, Amjad Almahairi, Bachman Philip, Eduard Hovy, and Aaron Courville. Calibrating energy-based generative adversarial networks. *ICLR submission*, 2017.

Yann N Dauphin, Razvan Pascanu, Caglar Gulcehre, Kyunghyun Cho, Surya Ganguli, and Yoshua Bengio. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. In *Advances in neural information processing systems*, pages 2933–2941, 2014.

Bert De Brabandere, Xu Jia, Tinne Tuytelaars, and Luc Van Gool. Dynamic filter networks. In *Neural Information Processing Systems (NIPS)*, 2016.

Vincent De Ladurantaye, Jacques Vanden-Abeele, and Jean Rouat. *Models of information processing in the visual cortex*. Citeseer, 2012.

Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Andrew Senior, Paul Tucker, Ke Yang, Quoc V Le, et al. Large scale distributed deep networks. In *Advances in neural information processing systems*, pages 1223–1231, 2012.

Olivier Delalleau and Yoshua Bengio. Shallow vs. deep sum-product networks. In *Advances in Neural Information Processing Systems*, pages 666–674, 2011.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. IEEE, 2009.

Misha Denil, Loris Bazzani, Hugo Larochelle, and Nando de Freitas. Learning where to attend with deep architectures for image tracking. *Neural computation*, 24(8):2151–2184, 2012.

Jean-Pierre Didier and Emmanuel Bigand. *Rethinking physical and rehabilitation medicine: New technologies induce new learning strategies*. Springer Science & Business Media, 2011.

Carl Doersch. Tutorial on variational autoencoders. *arXiv preprint arXiv:1606.05908*, 2016.

John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul): 2121–2159, 2011.

Angela Lee Duckworth, Eli Tsukayama, and Henry May. Establishing causality using longitudinal hierarchical linear modeling: An illustration predicting achievement from self-control. *Social psychological and personality science*, 2010.

Samuel Frederick Edwards and Phil W Anderson. Theory of spin glasses. *Journal of Physics F: Metal Physics*, 5(5):965, 1975.

Ronen Eldan and Ohad Shamir. The power of depth for feedforward neural networks. *arXiv preprint arXiv:1512.03965*, 2015.

Jeffrey L Elman. Finding structure in time. *Cognitive science*, 14(2):179–211, 1990.

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(Feb):625–660, 2010.

Scott E Fahlman and Christian Lebiere. The cascade-correlation learning architecture. 1989.

Marcus Frean. The upstart algorithm: A method for constructing and training feedforward neural networks. *Neural computation*, 2(2):198–209, 1990.

Kunihiko Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological cybernetics*, 36(4): 193–202, 1980.

Leon A Gatys, Alexander S Ecker, and Matthias Bethge. A neural algorithm of artistic style. *arXiv preprint arXiv:1508.06576*, 2015.

Tom Germano. Self organizing maps. *Available in http://davis. wpi. edu/˜ matt/courses/soms*, 1999.

Felix A Gers and Jürgen Schmidhuber. Recurrent nets that time and count. In *Neural Networks, 2000. IJCNN 2000, Proceedings of the IEEE-INNS-ENNS International Joint Conference on*, volume 3, pages 189–194. IEEE, 2000.

Ross Girshick. Fast r-cnn. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1440–1448, 2015.

Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.

Ian Goodfellow. Nips 2016 tutorial: Generative adversarial networks. *arXiv preprint arXiv:1701.00160*, 2016.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems*, pages 2672–2680, 2014.

Marco Gori and Alberto Tesi. On the problem of local minima in backpropagation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(1):76–86, 1992.

Céline Gravelines. *Deep Learning via Stacked Sparse Autoencoders for Automated Voxel-Wise Brain Parcellation Based on Functional Connectivity*. PhD thesis, The University of Western Ontario, 1991.

Alex Graves, Navdeep Jaitly, and Abdel-rahman Mohamed. Hybrid speech recognition with deep bidirectional lstm. In *Automatic Speech Recognition and Understanding (ASRU), 2013 IEEE Workshop on*, pages 273–278. IEEE, 2013.

Klaus Greff, Rupesh Kumar Srivastava, Jan Koutník, Bas R Steunebrink, and Jürgen Schmidhuber. Lstm: A search space odyssey. *arXiv preprint arXiv:1503.04069*, 2015.

Richard Gregory and Patrick Cavanagh. The blind spot. *Scholarpedia*, 6(10):9618, 2011.

Stephen Grossberg. Recurrent neural networks. *Scholarpedia*, 8(2):1888, 2013.

Aman Gupta, Haohan Wang, and Madhavi Ganapathiraju. Learning structure in gene expression data using deep architectures, with an application to gene clustering. In *Bioinformatics and Biomedicine (BIBM), 2015 IEEE International Conference on*, pages 1328–1335. IEEE, 2015.

Kevin Gurney. *An introduction to neural networks*. CRC press, 1997.

Shyam M Guthikonda. Kohonen self-organizing maps. *Wittenberg University*, 2005.

Bharath Hariharan, Pablo Arbeláez, Ross Girshick, and Jitendra Malik. Hypercolumns for object segmentation and fine-grained localization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 447–456, 2015.

David Hartley. *Observations on Man*, volume 1. Cambridge University Press, 2013.

Johan Hastad. Almost optimal lower bounds for small depth circuits. In *Proceedings of the eighteenth annual ACM symposium on Theory of computing*, pages 6–20. ACM, 1986.

James V Haxby, Elizabeth A Hoffman, and M Ida Gobbini. The distributed human neural system for face perception. *Trends in cognitive sciences*, 4(6):223–233, 2000.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*, 2015.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. *arXiv preprint arXiv:1603.05027*, 2016.

Donald Olding Hebb. *The organization of behavior: A neuropsychological theory*. Psychology Press, 1949.

Robert Hecht-Nielsen. Theory of the backpropagation neural network. In *Neural Networks, 1989. IJCNN., International Joint Conference on*, pages 593–605. IEEE, 1989.

Geoffrey E Hinton. Training products of experts by minimizing contrastive divergence. *Neural computation*, 14(8):1771–1800, 2002.

Geoffrey E Hinton, Peter Dayan, Brendan J Frey, and Radford M Neal. The" wake-sleep" algorithm for unsupervised neural networks. *Science*, 268(5214):1158, 1995.

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

Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

John J Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the national academy of sciences*, 79(8):2554–2558, 1982.

Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

Zhiting Hu, Xuezhe Ma, Zhengzhong Liu, Eduard Hovy, and Eric Xing. Harnessing deep neural networks with logic rules. *arXiv preprint arXiv:1603.06318*, 2016.

David H Hubel and Torsten N Wiesel. Receptive fields of single neurones in the cat's striate cortex. *The Journal of physiology*, 148(3):574–591, 1959.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

Michael I Jordan. Serial order: A parallel distributed processing approach. *Advances in psychology*, 121:471–495, 1986.

Nal Kalchbrenner, Aaron van den Oord, Karen Simonyan, Ivo Danihelka, Oriol Vinyals, Alex Graves, and Koray Kavukcuoglu. Video pixel networks. *arXiv preprint arXiv:1610.00527*, 2016.

Kiyoshi Kawaguchi. A multithreaded software model for backpropagation neural network applications. 2000.

AG Khachaturyan, SV Semenovskaya, and B Vainstein. A statistical-thermodynamic approach to determination of structure amplitude phases. *Sov. Phys. Crystallogr*, 24:519–524, 1979.

Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Diederik P Kingma, Shakir Mohamed, Danilo Jimenez Rezende, and Max Welling. Semi-supervised learning with deep generative models. In *Advances in Neural Information Processing Systems*, pages 3581–3589, 2014.

Tinne Hoff Kjeldsen. John von neumann's conception of the minimax theorem: a journey through different mathematical contexts. *Archive for history of exact sciences*, 56(1): 39–68, 2001.

Teuvo Kohonen. The self-organizing map. *Proceedings of the IEEE*, 78(9):1464–1480, 1990.

Bryan Kolb, Ian Q Whishaw, and G Campbell Teskey. *An introduction to brain and behavior*, volume 1273. 2014.

Mark L Krieg. A tutorial on bayesian belief networks. 2001.

Ranjay Krishna, Yuke Zhu, Oliver Groth, Justin Johnson, Kenji Hata, Joshua Kravitz, Stephanie Chen, Yannis Kalantidis, Li-Jia Li, David A Shamma, et al. Visual genome: Connecting language and vision using crowdsourced dense image annotations. *arXiv preprint arXiv:1602.07332*, 2016.

Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. 2009.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

Tejas D Kulkarni, William F Whitney, Pushmeet Kohli, and Josh Tenenbaum. Deep convolutional inverse graphics network. In *Advances in Neural Information Processing Systems*, pages 2539–2547, 2015.

Brenden M Lake, Ruslan Salakhutdinov, and Joshua B Tenenbaum. Human-level concept learning through probabilistic program induction. *Science*, 350(6266):1332–1338, 2015.

Alan S Lapedes and Robert M Farber. How neural nets work. In *Neural information processing systems*, pages 442–456, 1988.

Hugo Larochelle and Geoffrey E Hinton. Learning to combine foveal glimpses with a third-order boltzmann machine. In *Advances in neural information processing systems*, pages 1243–1251, 2010.

B Boser Le Cun, John S Denker, D Henderson, Richard E Howard, W Hubbard, and Lawrence D Jackel. Handwritten digit recognition with a back-propagation network. In *Advances in neural information processing systems*. Citeseer, 1990.

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

Yann LeCun, Corinna Cortes, and Christopher JC Burges. The mnist database of handwritten digits, 1998b.

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553): 436–444, 2015.

Honglak Lee, Roger Grosse, Rajesh Ranganath, and Andrew Y Ng. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In *Proceedings of the 26th annual international conference on machine learning*, pages 609–616. ACM, 2009.

Zhaoping Li. A neural model of contour integration in the primary visual cortex. *Neural computation*, 10(4):903–940, 1998.

Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European Conference on Computer Vision*, pages 740–755. Springer, 2014.

Cheng-Yuan Liou and Shiao-Lin Lin. Finite memory loading in hairy neurons. *Natural Computing*, 5(1):15–42, 2006.

Cheng-Yuan Liou and Shao-Kuo Yuan. Error tolerant associative memory. *Biological Cybernetics*, 81(4):331–342, 1999.

Christoph Lippert, Jennifer Listgarten, Ying Liu, Carl M Kadie, Robert I Davidson, and David Heckerman. Fast linear mixed models for genome-wide association studies. *Nature methods*, 8(10):833–835, 2011.

Zachary C Lipton, John Berkowitz, and Charles Elkan. A critical review of recurrent neural networks for sequence learning. *arXiv preprint arXiv:1506.00019*, 2015.

David G Lowe. Object recognition from local scale-invariant features. In *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*, volume 2, pages 1150–1157. Ieee, 1999.

Xuezhe Ma and Eduard Hovy. End-to-end sequence labeling via bi-directional lstm-cnns-crf. *arXiv preprint arXiv:1603.01354*, 2016.

Xuezhe Ma, Yingkai Gao, Zhiting Hu, Yaoliang Yu, Yuntian Deng, and Eduard Hovy. Dropout with expectation-linear regularization. *arXiv preprint arXiv:1609.08017*, 2016.

M Maschler, Eilon Solan, and Shmuel Zamir. Game theory. translated from the hebrew by ziv hellman and edited by mike borns, 2013.

Charles E McCulloch and John M Neuhaus. *Generalized linear mixed models*. Wiley Online Library, 2001.

Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

Marc Mézard and Jean-P Nadal. Learning in feedforward layered networks: The tiling algorithm. *Journal of Physics A: Mathematical and General*, 22(12):2191, 1989.

Marc Mézard, Giorgio Parisi, and Miguel-Angel Virasoro. Spin glass theory and beyond. 1990.

Marvin L Minski and Seymour A Papert. Perceptrons: an introduction to computational geometry. *MA: MIT Press, Cambridge*, 1969.

Melanie Mitchell. *An introduction to genetic algorithms*. MIT press, 1998.

Tom M Mitchell et al. Machine learning. wcb, 1997.

Jeffrey Moran and Robert Desimone. Selective attention gates visual processing in the extrastriate cortex. *Frontiers in cognitive neuroscience*, 229:342–345, 1985.

Michael C Mozer. A focused back-propagation algorithm for temporal pattern recognition. *Complex systems*, 3(4):349–381, 1989.

Kevin P Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012.

Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 807–814, 2010.

John Nash. Non-cooperative games. *Annals of mathematics*, pages 286–295, 1951.

John F Nash et al. Equilibrium points in n-person games. *Proc. Nat. Acad. Sci. USA*, 36 (1):48–49, 1950.

Anh Nguyen, Jason Yosinski, and Jeff Clune. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 427–436. IEEE, 2015.

Danh V Nguyen, Damla Şentürk, and Raymond J Carroll. Covariate-adjusted linear mixed effects model with an application to longitudinal data. *Journal of nonparametric statistics*, 20(6):459–481, 2008.

Erkki Oja. Simplified neuron model as a principal component analyzer. *Journal of mathematical biology*, 15(3):267–273, 1982.

Erkki Oja and Juha Karhunen. On stochastic approximation of the eigenvectors and eigenvalues of the expectation of a random matrix. *Journal of mathematical analysis and applications*, 106(1):69–84, 1985.

Aaron van den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. Pixel recurrent neural networks. *arXiv preprint arXiv:1601.06759*, 2016.

Keiichi Osako, Rita Singh, and Bhiksha Raj. Complex recurrent neural networks for denoising speech signals. In *Applications of Signal Processing to Audio and Acoustics (WASPAA), 2015 IEEE Workshop on*, pages 1–5. IEEE, 2015.

Rajesh G Parekh, Jihoon Yang, and Vasant Honavar. Constructive neural network learning algorithms for multi-category real-valued pattern classification. 1997.

Razvan Pascanu, Caglar Gulcehre, Kyunghyun Cho, and Yoshua Bengio. How to construct deep recurrent neural networks. *arXiv preprint arXiv:1312.6026*, 2013a.

Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. *ICML (3)*, 28:1310–1318, 2013b.

Razvan Pascanu, Yann N Dauphin, Surya Ganguli, and Yoshua Bengio. On the saddle point problem for non-convex optimization. *arXiv preprint arXiv:1405.4604*, 2014.

Bryan A Plummer, Liwei Wang, Chris M Cervantes, Juan C Caicedo, Julia Hockenmaier, and Svetlana Lazebnik. Flickr30k entities: Collecting region-to-phrase correspondences for richer image-to-sentence models. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2641–2649, 2015.

Tomaso Poggio and Thomas Serre. Models of visual cortex. *Scholarpedia*, 8(4):3516, 2013.

Christopher Poultney, Sumit Chopra, Yann L Cun, et al. Efficient learning of sparse representations with an energy-based model. In *Advances in neural information processing systems*, pages 1137–1144, 2006.

Jose C Principe, Neil R Euliano, and W Curt Lefebvre. *Neural and adaptive systems: fundamentals through simulations with CD-ROM*. John Wiley & Sons, Inc., 1999.

Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

Martin Riedmiller and Heinrich Braun. A direct adaptive method for faster backpropagation learning: The rprop algorithm. In *Neural Networks, 1993., IEEE International Conference On*, pages 586–591. IEEE, 1993.

AJ Robinson and Frank Fallside. *The utility driven dynamic error propagation network*. University of Cambridge Department of Engineering, 1987.

Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation. Technical report, DTIC Document, 1985.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3): 211–252, 2015.

S Rasoul Safavian and David Landgrebe. A survey of decision tree classifier methodology. 1990.

Ruslan Salakhutdinov and Geoffrey E Hinton. Deep boltzmann machines. In *AISTATS*, volume 1, page 3, 2009.

Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. In *Advances in Neural Information Processing Systems*, pages 2226–2234, 2016.

Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, 2015.

Mike Schuster and Kuldip K Paliwal. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681, 1997.

Thomas Serre, Lior Wolf, and Tomaso Poggio. Object recognition with features inspired by visual cortex. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 2, pages 994–1000. IEEE, 2005.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538*, 2017.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Paul Smolensky. Information processing in dynamical systems: Foundations of harmony theory. Technical report, DTIC Document, 1986.

Kihyuk Sohn, Honglak Lee, and Xinchen Yan. Learning structured output representation using deep conditional generative models. In *Advances in Neural Information Processing Systems*, pages 3483–3491, 2015.

Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

Amos Storkey. Increasing the capacity of a hopfield network without sacrificing functionality. In *International Conference on Artificial Neural Networks*, pages 451–456. Springer, 1997.

Richard S Sutton, David A McAllester, Satinder P Singh, Yishay Mansour, et al. Policy gradient methods for reinforcement learning with function approximation. In *NIPS*, volume 99, pages 1057–1063, 1999.

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–9, 2015.

Aaron van den Oord, Nal Kalchbrenner, Lasse Espeholt, Oriol Vinyals, Alex Graves, et al. Conditional image generation with pixelcnn decoders. In *Advances In Neural Information Processing Systems*, pages 4790–4798, 2016.

Andreas Veit, Michael J Wilber, and Serge Belongie. Residual networks behave like ensembles of relatively shallow networks. In *Advances in Neural Information Processing Systems*, pages 550–558, 2016.

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. *Journal of Machine Learning Research*, 11(Dec):3371–3408, 2010.

Martin J Wainwright, Michael I Jordan, et al. Graphical models, exponential families, and variational inference. *Foundations and Trends® in Machine Learning*, 1(1–2):1–305, 2008.

Brian A Wandell. *Foundations of vision.* Sinauer Associates, 1995.

Haohan Wang and Jingkang Yang. Multiple confounders correction with regularized linear mixed effect models, with application in biological processes. 2016.

Haohan Wang, Aaksha Meghawat, Louis-Philippe Morency, and Eric P Xing. Select-additive learning: Improving cross-individual generalization in multimodal sentiment analysis. *arXiv preprint arXiv:1609.05244*, 2016.

Paul J Werbos. Generalization of backpropagation with application to a recurrent gas market model. *Neural Networks*, 1(4):339–356, 1988.

Paul J Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.

Bernard Widrow et al. *Adaptive" adaline" Neuron Using Chemical" memistors.".* 1960.

Alan L Wilkes and Nicholas J Wade. Bain on neural networks. *Brain and cognition*, 33(3): 295–305, 1997.

Gibbs J Willard. Elementary principles in statistical mechanics. *The Rational Foundation of Thermodynamics, New York, Charles Scribners sons and London, Edward Arnold*, 1902.

Andrew G Wilson, Christoph Dann, Chris Lucas, and Eric P Xing. The human kernel. In *Advances in Neural Information Processing Systems*, pages 2854–2862, 2015.

SHI Xingjian, Zhourong Chen, Hao Wang, Dit-Yan Yeung, Wai-Kin Wong, and Wang-chun Woo. Convolutional lstm network: A machine learning approach for precipitation nowcasting. In *Advances in Neural Information Processing Systems*, pages 802–810, 2015.

Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhutdinov, Richard S Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. *arXiv preprint arXiv:1502.03044*, 2(3):5, 2015.

Zhilin Yang, Ruslan Salakhutdinov, and William Cohen. Multi-task cross-lingual sequence tagging from scratch. *arXiv preprint arXiv:1603.06270*, 2016.

Andrew Chi-Chih Yao. Separating the polynomial-time hierarchy by oracles. In *26th Annual Symposium on Foundations of Computer Science (sfcs 1985)*, 1985.

Xin Yao. Evolving artificial neural networks. *Proceedings of the IEEE*, 87(9):1423–1447, 1999.

Dong Yu, Li Deng, and George Dahl. Roles of pre-training and fine-tuning in context-dependent dbn-hmms for real-world speech recognition. In *Proc. NIPS Workshop on Deep Learning and Unsupervised Feature Learning*, 2010.

Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016.

Matthew D Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. *arXiv preprint arXiv:1611.03530*, 2016a.

Ke Zhang, Miao Sun, Tony X Han, Xingfang Yuan, Liru Guo, and Tao Liu. Residual networks of residual networks: Multilevel residual networks. *arXiv preprint arXiv:1608.02908*, 2016b.

Junbo Zhao, Michael Mathieu, and Yann LeCun. Energy-based generative adversarial network. *arXiv preprint arXiv:1609.03126*, 2016.

Xiang Zhou and Matthew Stephens. Genome-wide efficient mixed-model analysis for association studies. *Nature genetics*, 44(7):821–824, 2012.