This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/OJCAS.2020.3032092, IEEE Open Journal of Circuits and Systems

Brain-Inspired Computing: Models and Architectures

Invited Paper

Keshab K. Parhi, *Fellow, IEEE*; and Nanda K. Unnikrishnan, *Student Member, IEEE*

With an exponential increase in the amount of data collected per day, the fields of artificial intelligence and machine learning continue to progress at a rapid pace with respect to algorithms, models, applications, and hardware. In particular, deep neural networks have revolutionized these fields by providing unprecedented human-like performance in solving many real-world problems such as image or speech recognition. There is also significant research aimed at unraveling the principles of computation in large biological neural networks and, in particular, biologically plausible spiking neural networks. This paper presents an overview of the *brain-inspired* computing models starting with the development of the perceptron and multi-layer perceptron followed by convolutional neural networks (CNNs) and recurrent neural networks (RNNs). The paper also briefly reviews other neural network models such as Hopfield neural networks and Boltzmann machines. Other models such as spiking neural networks (SNNs) and hyperdimensional computing are then briefly reviewed. Recent advances in these neural networks and graph related neural networks are then described.

***Index Terms*—Perceptron, Multi-Layer Perceptron, Convolutional Neural Network, Recurrent Neural Network, Hopfield Neural Network, Boltzmann Machines, Hyperdimensional Computing, Spiking Neural Networks, Graph Neural Networks**

I. INTRODUCTION

MACHINE learning and data analytics continue to expand the fourth industrial revolution and affect many aspects of our daily lives. At the heart of this drive is the quest for artificial intelligence (AI) and for design of machines that can learn. This quest has advanced speech processing to an extent that voice assistants have become ubiquitous in our households [1]. Computer vision based application are now capable of reaching super human levels at tasks such as image classification [2], [3], [4]. Even games that were once thought unwinnable like Go or starcraft, machines have learned to outplay even the best humans [5], [6]. Machines that can learn also have had the significant impact in the field of medical diagnostics where deep learning has been used to identify diseases like diabetic retinopathy [7], gastrointestinal bleeding [8], and cardiovascular diseases [9].

Looking back at the last decade, deep learning has pushed the limits of machine capabilities for inference in edge devices. This advance can be attributed to the availability of abundant high-quality data [10], [11] and availability of accelerators and GPUs that enable faster training times. With further advancement in systems for deep learning, frameworks like TensorFlow or PyTorch have been developed [12], [13] to enable ready access. These tools have greatly facilitated *transfer learning* where pre-trained models learned from one dataset can be refined to learn models of another dataset. This reduces training time and improves accuracy. Due to the democratization of the tools and their availability, and transfer learning, deep learning as a research tool is now available to everyone and consumers can reap the benefits by using their edge devices.

This work has been supported in part by the National Science Foundation under Grants CCF-1814759 and CCF-1954749.

K. K Parhi and N. K. Unnikrishnan are with the department of Electrical and Computer engineering at the University of Minnesota, Minneapolis, MN 55455 USA (e-mail: parhi@umn.edu, unnik005@umn.edu).

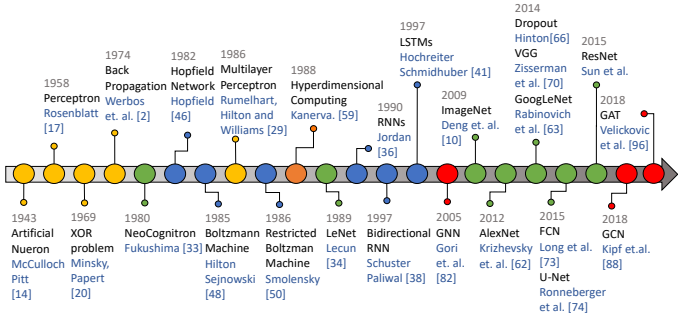


Fig. 1. Evolution of neural network models.

Though deep learning and AI appear new and revolutionary, these have a storied and evolutionary past. These models evolved over many decades and were inspired by the human brain. The reader is referred to [14] for a zoo of neural networks. The human brain has about 100 billion neurons and 1000 trillion synapses. Therefore, the brain can be described by a sparse network. The spiking operations in the human brain consume about 10 Watts and achieve an energy efficiency of 10 Top/s/W with an average output spiking rate of 1 Hz [15]. The human brain consumes significantly less energy compared to computers and is highly energy-efficient. Thus, there is a desire to create models that mimic the brain. This paper describes the various brain-inspired models and the historical context in which these models were developed. This historical context lends a new perspective about what the limitations of the existing models were and how these were overcome by the new models.

To provide this historical context, this paper starts with how understanding the neurons in the brain led to the development of one of the first neural network models, the McCulloch-Pitts neuron, and its ability to represent a basic logic reasoning. The subsequent perceptron model was inspired by Hebbian learning; this model introduced the concept of learning and

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/OJCAS.2020.3032092, IEEE Open Journal of Circuits and Systems

adaptation. Various limitations of the perceptron led to a pause in AI research; however, this also led to the multilayer prerceptrons (MLPs) that overcame the limitations. It was shown that the MLPs can act as universal approximators. One of the most consequential advance was the development of efficient backpropagation algorithms that are used to learn the parameters in most neural network models today. Similarly, better understanding of vision led to the development of one of the first vision based models, the Neocognitron. This first generation Neocognitron model served as the baseline for the development of the convolutional neural network (CNN) architecture. The recurrent neural networks (RNN) and long short-term memory (LSTM) were developed to exploit the temporal properties of time-series data and are used for speech and language processing. Finally, biologically plausible spiking neural network (SNN) can be seen as the third wave of neural networks that are inspired by the brain and and try to replicate the extreme energy-efficiency of the brain.

This paper is organized as follows. Section II describes the evolution of the various early brain-inspired computing models ranging from perceptron and MLP to Neocognitron. This section also describes the Hopfield neural network and Boltzmann machines, followed by biologically inspired neurons and the hyperdimensional computing model. The evolution of different models is illustrated in the chronological chart shown in Fig. 1. Section III introduces deep neural network models such as CNN, RNN and SNN. Section IV describes recent advances including deep convolutional neural networks (DNNs) and graph neural networks.

II. Evolution of Early Models

A. The First Mathematical Neuron Model

McCulloch, a neuropsychologist, and Pitts, a logician, together developed the first known mathematical model of a biological neuron in 1943 [16]. The McCulloch-Pitts Neuron (MPN), as it came to be known, is shown in Fig. 2. The model was defined with the following assumptions of acyclic nets.

- The activity of the neuron is an "all-or-none" process.
- A certain fixed number of synapses must be excited within the period of latent addition in order to excite a neuron at any time, and this number is independent of previous activity and position on the neuron.
- The only significant delay within the nervous system is the synaptic delay.
- The activity of any inhibitory synapse absolutely prevents the excitation of the neuron at that time.
- The structure of the net does not change with time.

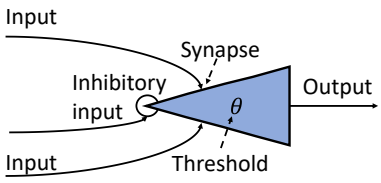


Fig. 2. Representation of a McCulloch-Pitts neuron as seen in [16].

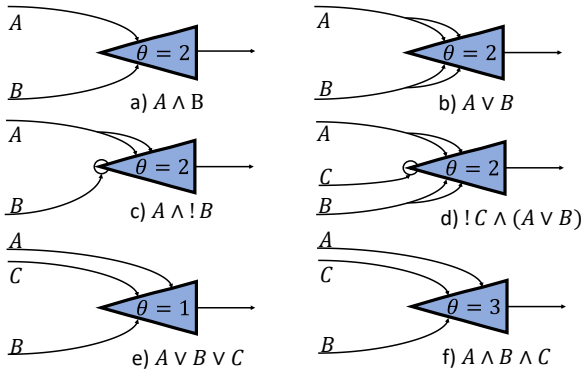


Fig. 3. Implementation of different Boolean logic gates with a McCulloch-Pitts neuron as seen in [16]. The neurons marked with bubbles are inhibitory neurons.

The MPN model is mathematically equivalent to a linear threshold unit. The MPN is based on binary logic, and its operation can be described using two phases. In the first phase, the neuron sums the excitations from all upstream neurons. In the second phase, the neuron fires only if a predefined threshold, θ, is met, and no inhibitory neuron has fired. This simple thresholding logic was quite compelling and could be used to simulate a variety of Boolean functions, as shown in Fig. 3, designed in the original style as described by [16].

The functionality of the network can be easily understood by examining each gate individually. For example, Fig. 3(a) shows the functionality of an *AND* logic gate. The inputs A and B are connected to the neuron via synapses. When an input is excited, i.e., when the neuron driving it fires, it is represented by the Boolean logic value of 1; otherwise its logic value is 0. For this neuron to fire, a minimum number of input synapses, as defined by the threshold variable θ, must be excited. In the AND gate as $\theta = 2$, both inputs must be excited for the neuron to fire. This logic can be extended to the Boolean *OR* operation seen in Fig. 3(b). For the threshold $\theta = 2$, to implement the OR Boolean logic, each input has at least θ connections to the neuron, although that need not be the case. Thus far, only the use of non-inhibitory inputs was shown; however, one of the main characteristics of the MPN was the introduction of inhibitory inputs. An inhibitory input is defined such that if it is excited, then the neuron will not fire irrespective of the state of all other inputs. This logic is illustrated in Fig. 3(c), where the input B acts as an inhibitory input. If B is excited, then the neuron will not fire even if A is excited. If B is inactive, the output of the neuron depends on the state of A. Thus, the inhibitory input allows for the construction of Boolean inversion logic; in this case, the neuron implements the function $A \bigwedge !B$, where ! represents the logical inversion function. Fig. 4 shows the graphical representation for 2 and 3 input AND and OR gates with the decision separation boundary.

Thus we see that the MPN can be adapted to represent a variety of Boolean functions. Revolutionary as this model was, it did suffer from a set of shortcomings as listed below:

- Representations limited to logical inputs.
- Functions need to be created and defined manually.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/OJCAS.2020.3032092, IEEE Open Journal of Circuits and Systems

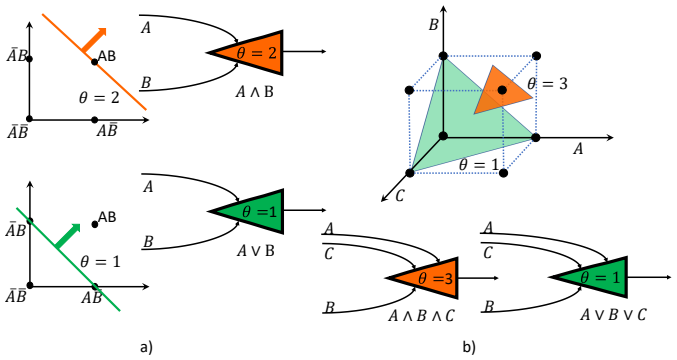


Fig. 4. Graphical representation of MPN for *AND* gate and *OR* gate. a) 2-input neurons. b) 3-input neurons.

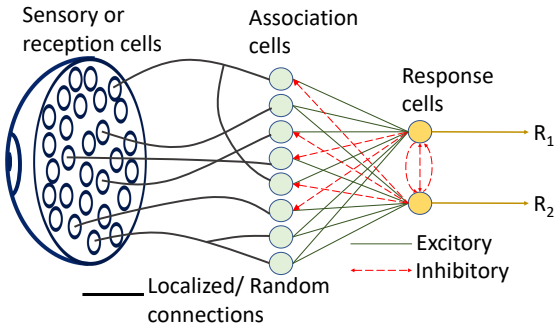


Fig. 5. Visualization of the perceptron model, taken from Fig. 2A of [19].

- It does not support any notion of learning.
- Functions need to be linearly separable.
- Equal weight for all the inputs.

These limitations prevent its use in real-world applications where the values are continuous and the data are often not linearly separable. The equal weight constraint limits the number of functions that can be represented and necessitates the need for more complicated models where different inputs can be assigned different weights. This model cannot accommodate negative or fractional weights. Finally, the model cannot learn an optimal weight function like a biological neuron.

B. The Perceptron and the Ability to Learn

The key drawback of the MPN is that the neurons cannot be trained to *learn* the weights of the model from the input data. Thus further developments in the field were inspired by how biological neurons learn the underlying functions. Hebb hypothesized in 1949 how neurons are connected in the brain: "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 A's efficiency, as one of the cells firing B, is increased." [17].

This was one of the most iconic statements that summarized the development of the *Hebb Synapse* [18]. A significant takeaway was the concept of the *growth process* of neurons to increase the efficiency of the cells. This idea was instrumental in understanding how neurons learned, and it inspired Rosenblatt's development of the *perceptron* in 1958. The perceptron was designed to be a more generalized neuron and is a foundation of modern deep learning systems.

Fig. 5 describes the structure of a single-layer perceptron. The structure consists of three types of cells; the first type is the sensory or reception cells (S_j) that capture the input. The second type of cells correspond to the association cells (A_j) that weigh the input features before passing them to the third type of cells, the response cells (R_j). The association cells are connected to the sensory inputs via localized/focalized or random connections. An association cell's inputs are referred to as origin points, and these connections may be excitatory or inhibitory such that an association cell *fires* or is *active* when the algebraic sum of all its inputs is greater than the threshold θ. Response cells are similar to the association cells and often have a large set of cells as origin points called its *source set*. The connections between the sensory cells and the association cells are feedforward and fixed. However, the connections from the association cells to the response cells have feedback paths allowing to alter the intensity of the specific connections.

These connections between the association cells and response cells in the perceptron can have excitatory feedback paths to synapses from its source set that reinforce the connections in its source set that led to the required response. Thereby this serves as a mechanism to make the same response occurring more likely if presented with the same input. Similarly, it can have inhibitory feedback paths to cells that complement its source set, or to other responses, to hinder the occurrence of a complementary response as shown by the red arrows in Fig. 5. Thus, the development of perceptron was targeted to overcome some of the shortcomings of the MPN. One of the fundamental changes was the abolition of the uniform weighting of all input synapses. While the inputs in the MPN were restricted to logical values, the perceptron has no such limitation. Thus, the inputs in the perceptron can be associated with weights where these weights can be trained. This also allows for the implicit definition of inhibitory inputs rather than a dedicated input. These can all be put together to define the training algorithm for the perceptron as shown in Algorithm 1.

A perceptron can be trained by providing a stimulus or *learning series* and recording its responses. If the response matches the system's desired response, there is a positive reinforcement of the weights when the response is correct and negative reinforcement when it is incorrect. The learning series can consist of thousands of individual stimuli. Once the *learning phase* completes, the weights are fixed, and the perceptron can either be evaluated by the same learning series or by a new unseen series. In the first case, the same learning series is fed to the perceptron, where the learning phase would have biased the output to the correct response. In the second case, a completely new and previously unseen stimulus from the same distribution is fed to the perceptron to gauge its performance. In both cases, when operating on *differentiating models* as the size of the stimuli increases, the probability that the evaluation set predicts the correct response increases and converges to 1 for completely separable data.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/OJCAS.2020.3032092, IEEE Open Journal of Circuits and Systems

Algorithm 1 Perceptron training algorithm for a simple reinforcement based γ system [20].

Input: *Training set: Sensory inputs* S^i, *Responses* R^i
Network Variables: *Association Cells:* A_j
while *not converged* **do**
  for i *in training set,* $i \in \{1, \ldots, N\}$ **do**
    Drive Sensory inputs S^i
    Measure Response R^i
    for R^i_k, $k \in \{1, \ldots, K\}$ **do**
      if R^i_k matches desired Response $\hat{R}^i_k$ **then**
        for *All* A_j *in source-set* **do**
          Reinforce A_j
          Positive ΔV *is added to active* A_j
          Negative ΔV *is added to inactive* A_j
        end for
      else
        for *All incident* A_j *in source-set* **do**
          Inhibit A_j
          Negative ΔV *is added to active* A_j
          Positive ΔV *is added to inactive* A_j
        end for
      end if
    end for
  end for
end while

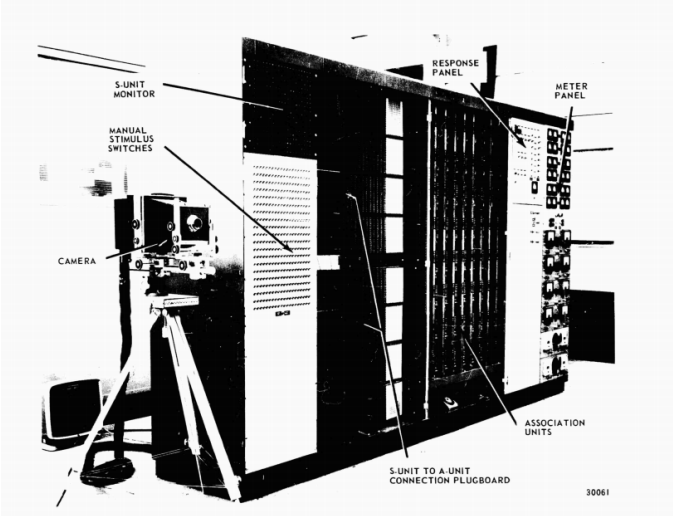


Fig. 6. Original Mark I perceptron as seen in its operator's manual [20].

Thus the perceptron training algorithm can learn patterns and associations from the statistical nature of the data.

The original Mark-I perceptron, as shown in Fig. 6[1], was the first hardware implementation of Fig. 5. The sensory input either was obtained from the camera or could be set by the sensory switch panel. Finally, the response of the system could be seen in the response panel. It made use of potentiometers to tune the different weights or association cells of the device. This machine was revolutionary at the time, leading to very high expectations from the press[2]:

> "Stories about the creation of machines having human qualities have long been a fascinating province in the realm of science fiction," Rosenblatt wrote in 1958."Yet we are about to witness the birth of such a machine – a machine capable of perceiving, recognizing, and identifying its surroundings without any human training or control." "The embryo of an electronic computer that [the Navy] expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence."

[1] A high-resolution image is available in the digital archives [21]
[2] "Electronic 'Brain' Teaches Itself." The New York Times, 13 July 1958.

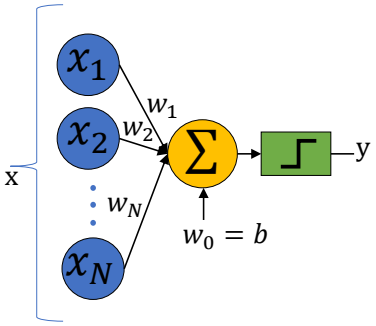


Fig. 7. Mathematical model of a perceptron.

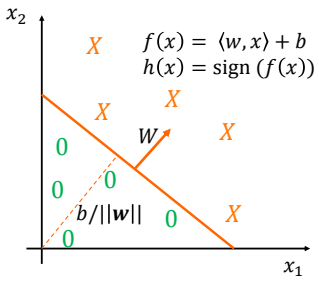


Fig. 8. Linear separation of classes with perceptron.

C. Limitation of a Single Perceptron

The perceptron model described thus far can be represented using a simpler mathematical model, as shown in Fig. 7. Mathematically, a perceptron can be modeled as Eq. (1)

$$y = \sigma(\sum_{k=1}^{N} w_k x_k + b) \tag{1}$$

where x_k represents the input from synapse k, W_k represents the weight of the synapse k and b represents the bias and can be interpreted as the negative of the threshold for firing θ defined earlier. σ represents a non linear activation function and y is the output of the neuron.

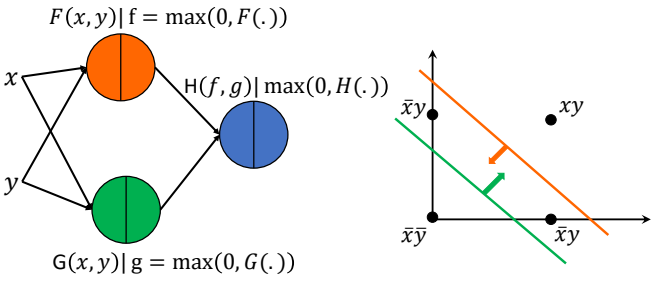


Fig. 9. The XOR problem, no linear separation boundary of single layer perceptron can separate the classes. The XOR problem can be solved by a multi-layer perceptron as shown on the left.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/OJCAS.2020.3032092, IEEE Open Journal of Circuits and Systems

This single perceptron is quite capable of performing any type of linear discrimination task, as illustrated in Fig. 8. However, the same advantage is one of the weaknesses of the model that a single-layer perceptron is unable to distinguish classes that are not linearly separable accurately. This was particularly highlighted by Minsky and Papert, mathematicians and computer scientists, in their 1969 book [22] characterizing the capabilities and limitations of loop-free learning and pattern recognition machines.

The perceptrons suffer from two limitations: the problems of *parity* and *connectedness*. To understand these limitations, we provide a simplistic definition of the terms provided in [22]. A *predicate* is a function that computes a Boolean result from a set of inputs. A family of *predicates* are chosen based on a linearly weighted threshold function taking a set of predicates as inputs. In Eq. (1), x represents the predicates and b represents the threshold. The perceptron model is defined as one that can represent any such predicate from this family. The order of the predicates is defined as the minimum number of input elements that any of internal predicate, x, depends on in order to compute the overall functionality [23]. These definitions can be used to examine the parity problem, deciphering whether a given set of data has an odd or even number of points. The parity predicate theorem postulates that the order of the predicate must be at least $|R|$, which is the size of the set R, a finite set of points that represent the entire input space or *retina*. An example to explain this concept is that of the *XOR problem*. Fig. 9 shows a visual representation of this problem where the first layer of 2 neurons represents the input predicates, and the final output layer represents a linearly weighted threshold predicate such as the perceptron. To fully represent the XOR problem, at least one of the input predicates must look at the entire input space. At least one association unit has to receive connections from the entire input space in the context of the perceptron. This example illustrates that the capability of the perceptron is limited.

The second limitation related to connectedness. Connectedness is defined as the set, where between two points in the set, there exists a connected path between them through any of the points in the set. Two points are considered connected if they are adjacent. Like the parity problem, the order of the predicate to solve this problem is proportional to the entire input space. Thus the main argument of the perceptron is that any algorithm for the perceptron is dependent on the size of the input space and changing the input size amounts to designing a new algorithm.

While the parity and connectedness showed the limits of the usefulness of the perceptron, it was argued that too much focus was on the limitation without sufficient merit given to its benefits [23], [24]. Also, the emphasis is made on the fact that the models studied are a severely limited class of model compared to the original intention of the design.

D. Multilayer Perceptrons

Multilayer perceptrons (MLPs) are a class of models that try to overcome some of the limitations by stacking layers of perceptrons one after another. Stacking of these nonlinear layers together allows for the network to learn complex relationships between the inputs and output. As MLPs were very good at approximating functions, it could be considered as a universal approximator [25]. The universal approximation theorem states that a feedforward network with a single hidden layer containing a finite number of neurons can approximate continuous functions on compact subsets of the real space, under certain assumptions on the activation function. The first mathematical proof of the universal approximation theorem applied to neural networks was shown with the sigmoid activation function [26].

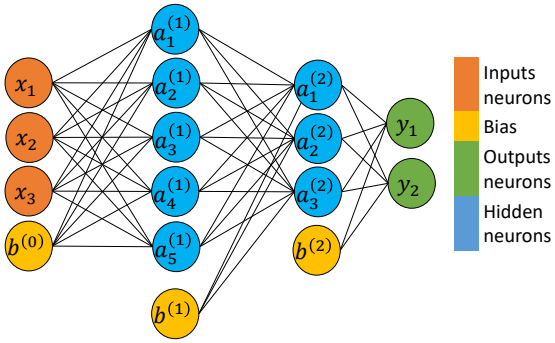


Fig. 10. Architecture of a simple 3-layer multilayer perceptron, with 3 input features and two output features.

The structure of an MLP is shown in Fig. 10, where the output of layer l is represented as $\boldsymbol{a}^{(l)}$. Special cases of the general layer are the input layer $\boldsymbol{a}^{(0)} = \boldsymbol{X}$ and output layer $\boldsymbol{Y} = \boldsymbol{a}^{(L)}$ for an L layer network. The inner operation of a multilayer perceptron is described in Eqs. (2) and (3)

$$\boldsymbol{z}^{(l)} = \boldsymbol{W}^{(l)}\boldsymbol{a}^{(l-1)} + \boldsymbol{b}^{(l)} \tag{2}$$

$$\boldsymbol{a}^{(l)} = \sigma(\boldsymbol{z}^{(l)}) \tag{3}$$

where $\boldsymbol{W}^{(l)}$, and $\boldsymbol{b}^{(l)}$ represent the learnable weights and bias for layer l, $\boldsymbol{z}^{(l)}$ is the matrix-vector multiplication of weights and the input activations, and $\boldsymbol{a}^{l}$ is the $d^{(l)}$-dimensional output of the non-linear activation function $\sigma(.)$ (e.g., sigmoid, tanh, ReLU). Here $d^{(l)}$ is the number of neurons in layer l. This can be extended as a matrix-matrix multiplication when working with mini-batches with the additional dimension N for the size of the mini-batch.

Supervised training of these MLP models is performed with the gradient descent algorithm minimizing the error in a complex parameter space that defines the neural network. The *backpropagation* algorithm was developed as an efficient way to propagate the gradients of the errors to the previous layers; a simplified version of this algorithm is based entirely on the chain rule [27], [28]. An efficient form of the backpropagation algorithm in neural network like objects is attributed to [29] and its first demonstrated use with neural networks is attributed to [30]. The backpropagation in its current form to optimize a neural network is based on and popularized by Rumelhart [31].

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/OJCAS.2020.3032092, IEEE Open Journal of Circuits and Systems

E. Evolution of the Vision Model

Thus far, the early efforts had attempted to model and replicate biological networks. One of the earliest applications of these models was image processing. However, these earlier models did not explicitly exploit the spatial aspect of the visual processing in the brain. One of the first studies of these aspects was conducted by Hubel and Wiesel, two neurophysiologists, in 1959. Their study was based on the fact that vision is a behavioral judgment in response to visual stimulation—it discusses how the brain organizes disconnected bits of information to whole objects by processing the images. The observations are based on experiments conducted on twenty-four cats by beaming lights on the eyes and recording the brain's response [32]. While projecting patterns of light and dark on a screen in front of the cat, they observed that specific neurons fired rapidly when presented with lines at one angle, while others neurons responded best to another. These studies elaborated on the visual system's capabilities to build an image from simple stimuli into more complex representations.

The restricted retina area was called a receptive field, and these were divided into excitatory or inhibitory fields. The observation was that any light stimulus covering the entire retina was ineffective. To obtain a clear pattern, light must fall on the excitatory fields and not on the inhibitory fields. The orientation of these receptive fields could be in any manner, including horizontal, vertical, and oblique orientations. A spot of light gave better response in some directions than others. The conclusion was that the oriented slits of light were the most effective method for activating the neurons in the brain.

Furthermore, in 1962, it was found that processing is done by two types of cells, simple S cells, and complex C cells [33]. Complex cells were developed, aggregating the information from multiple simple cells. The complex cell can be understood as a function that responded to the output from a bank of simple cells robust to distortion. With the progression of research, they were able to define more complex responses in their later papers. However, though this model was able to explain a lot of the vision's fundamental characteristics, it had its pitfalls. Most notably, these models did not address many essential features like color and spatial frequency. Nevertheless, artificial neural networks (ANNs), fundamental components of modern deep learning, owe their origin to the concept of cascading models of cell types inspired by Hubel and Wiesel's observations. In 1981, Hubel and Wiesel received the Nobel Prize for Physiology or Medicine, recognizing them for their research on the development and understanding of the visual system [34].

Fukushima, a computer scientist, observing some of the shortcomings in the Hubel and Wiesel model, subsequently developed the NeoCognitron model in 1980 [35]. The structure of the network has its inspiration from the visual system described by Hubel and Wiesel earlier. It consists of an input layer followed by several connections of modular structures, each of which consists of two layers of cells. The first layer of the module consists of simple cells or S cells, and the second layer, of complex cells or C cells. This was an early attempt at designing a neural network model with the same advanced capability for pattern recognition as humans. The S plane consists entirely of S cells and similarly the C plane of C cells.

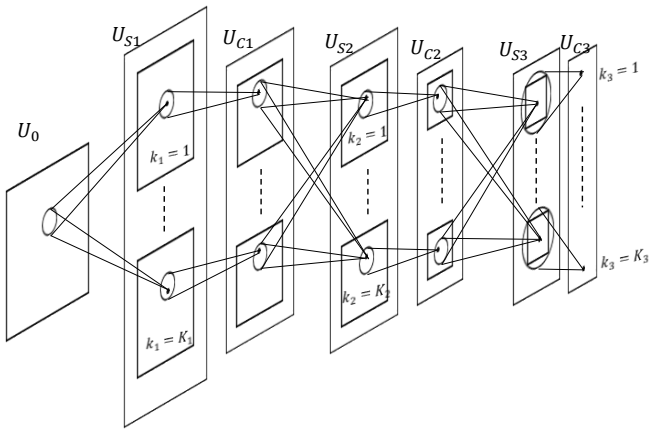


Fig. 11. Model of NeoCognitron illustrating the interconnections between layers [35].

Fig. 11 shows the NeoCognitron model highlighting the different types of layers and the interconnections between layers. Each tetragon drawn with heavy lines represents an S-plane or a C-plane, and each vertical tetragon drawn with thin lines, in which S-planes or C-planes are enclosed, represents an S-layer or a C-layer.

The S cells are more specialized and are designed to respond only to particular features. Each S cell in an S-plane has an identical response to its receptive field, thereby allowing adjacent cells to process slightly shifted version of the image. C cells are designed to complement S cells and handle the positional errors in the network. To enable this, every C cell has connections from a group of S cells at different positions. Thus this interleaving of S layers and C layers form the network of the system. After the learning phase, the NeoCognitron can recognize input patterns and is robust to changes in shape, size, or position.

There are two types of connections present in NeoCognitron. The first set of connections are learnable connections that connect the C layer to the next S layer. The goal of these connections is to learn the patterns from the previous layer. The second set of connections are fixed connections that connect the S layer to the next C layer. These connections do not learn and are designed to refine and confirm the learnings of the S layer. One of the consequences of the layered design is the ability of NeoCognitron to capture local and global features from the image adequately. The active area of the original image that a cell looks at increases with the depth of the network. This is referred to as the *receptive field* of the cell, and as the receptive field increases, it allows the network to develop more complex visual concepts. The effect of receptive fields is illustrated in Fig. 12.

NeoCognitron uses a Hebbian-based unsupervised learning approach where the updates to the learnable parameters are proportional to the inputs and outputs of the design. Within the S layer, only the S-plane with the maximum value is reinforced with the update, thereby allowing different planes to learn

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/OJCAS.2020.3032092, IEEE Open Journal of Circuits and Systems

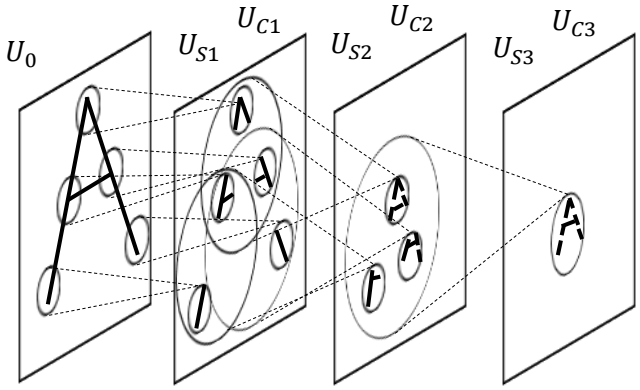


Fig. 12. Receptive field of each layer in NeoCognitron [35].

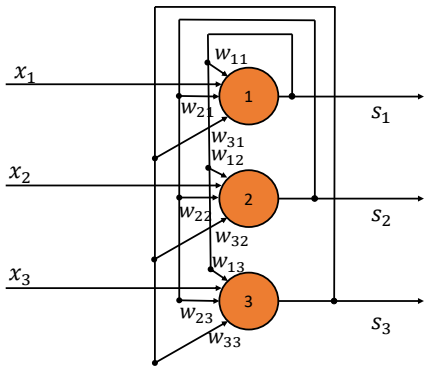


Fig. 13. Schematic representation of a Hopfield network adapted from [36].

different attributes of features. This unsupervised approach can be interpreted as a form of clustering of the target application.

F. Revisiting Hebbian Learning and Hopfield Networks

Hebbian learning theorized that repeated and persistent firing causes metabolic changes that reinforce the connection between neurons: "Neurons that fire together wire together." This stipulates that repeated patterns of synaptic activity create either a persistent increase or decrease in the synaptic efficacy of neurons. This, referred to as long-term potentiation (LTP) and long-term depression (LTD), underlines the concept of neuron synaptic plasticity. This is an explicit form of learning where a group of firing neurons sculpts an event in the hippocampus. An analogy of this can be understood with the following example adapted from [36]. Let the presence of dark clouds cause the firing of one set of neurons and the occurrence of rain cause the firing of a different set of neurons. When both events co-occur, both sets of neurons fire concurrently, thereby strengthening the synapses between them. Thus this implicitly defines memory in an associative manner that can retrieve elements by forming associations between its different characteristics.

Hopfield networks are a special kind of artificial neural networks, introduced by Hopfield in 1982 [37], that train, store and retrieve memory in a similar associative manner. This associative memory assumes the state of the brain is defined by neurons that fired recently [38]. The network can be trained such that it stores specific patterns or memories within the system. As it is an associative memory, it can retrieve its stored patterns even when provided with partial or corrupted inputs.

A Hopfield network is a type of fully connected recurrent neural network. It is composed of a simple neuron and has a high degree of similarity to the MPN and perceptron. Each neuron i can be in one of two possible states: the not-firing state or $s_i = 0$ or the firing state or $s_i = 1$. The weight of the connection from neuron i to neuron j is w_{ij}, which defines the synaptic strength of the connection. A neuron i will enter a firing state if the weighted sum of all of its inputs exceeds a predefined fixed threshold, θ_i, else it will remain in a not-firing state. Neurons evaluate their state asynchronously and randomly to check if they exceed the threshold, only ensuring that they maintain an average rate of evaluations (R). The structure of the Hopfield network is shown in Fig. 13. Hopfield networks were constrained so that the synaptic weight of self connections (w_{ii}) were set to zero, and all connections were symmetric ($w_{ij} = w_{ji}$), though the latter was not mandatory. The update mechanism for a neuron can be defined mathematically, as shown in Eq. (4).

$$s_i(k+1) = \begin{cases} 1, & \text{if } \sum_{j=0}^{N} w_{ij} s_j(k) \geq \theta_i \\ 0, & \text{otherwise} \end{cases} \tag{4}$$

$$E = -\frac{1}{2} \sum \sum_{i \neq j} w_{ij} s_i s_j \tag{5}$$

where $s_i(k)$ represents the current state of neuron i at time step k and N is the total number of neurons in the system. The total energy (E) of a given state is defined by Eq. (5). When the synaptic weights (w_{ij}) of the network have been trained with a particular state vector, the network will be in a stable equilibrium state that will define that state as a minimum energy point. Given a new input state, it can be observed that the system is set in a way that the system's energy is a monotonically decreasing function with respect to a change in s_i. Thus once the network has stable low energy points or local minima when given a particular input, it will move in a direction that minimizes its energy before finally stopping at a local minimum. Thus this is what gives Hopfield network the ability to accept partial or even corrupted inputs and move in the direction towards a local energy minimum.

Hopfield network model has three distinct differences with the prior models, such as MPN and perceptron. First, prior models such as the perceptrons were almost exclusively feed-forward. This meant that the flow of information was only in the forward direction with no feedback or *backward couplings*. Hopfield networks, however, rely heavily on backward couplings to form associative memories. Second, prior models work in an inference like manner interpreting data directly recognizing patterns without attempting to find out more abstract understanding of the patterns. Lastly, prior models were designed in a synchronous manner, which is not observed in biological processes and would also be very difficult to achieve in such processes.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/OJCAS.2020.3032092, IEEE Open Journal of Circuits and Systems

G. Boltzmann Machines

Boltzmann machines are a variation of Hopfield networks proposed by Hinton and Sejnowski in 1985 [39]. The critical difference is that rather than defining a fixed threshold, it defines a probabilistic model on the likelihood of the neuron to fire with a given set of inputs. The probability of a neuron firing is the sigmoid of the weighted sum of the inputs and its corresponding synaptic strengths.

$$P(s_i = 1) = \sigma(\sum_{j}^{N} w_{ij} s_j) \tag{6}$$

where σ represents the sigmoid function and $P(x)$ represents the probability of an event x occurring.

If the update order of processing the neurons follows any order but the total number of inputs, the network will follow a Boltzmann distribution where the probability of a given state vector, defined by each neuron's state, depends only on the energy of the state. The energy of a state vector follows the same definition, as seen in Hopfield networks in Eq. (5).

Therefore these systems are a class of non-deterministic or stochastic generative neural network models. Like the Hopfield network, Boltzmann machines are symmetric and fully connected without any self neuron loops. However, unlike Hopfield networks, it defines two types of neurons: visible neurons and hidden neurons.

Boltzmann machines can be used to solve two distinct problems [40]. First, it can be used to find the optimization of a cost function where the network's synaptic weights are fixed, and the network's energy is a representation of the cost function. Second, it can solve the converse problem for a given set of inputs; it finds the weight for which the inputs represent an excellent solution to its optimization problem.

For the learning problem, to train the network, the weights are adjusted to increase the probability, $P(s)$, that the state vector s is the optimal result, by performing a gradient ascent operation. The derivative of the log probability function is described by Eq. (7).

$$E_{data}\left[\frac{\partial \log P(s)}{\partial w_{ij}}\right] = E_{data}[s_i s_j] - E_{model}[s_i s_j] \tag{7}$$

where $E_{data}[x]$ represents the expected value of x in a data distribution, and $E_{model}[x]$ represents expected value of the model when sampling values in its equilibrium state. The learning rule follows a gradient ascent optimization, where the derivative is multiplied by some learning rate and added to the weight w_{ij}. Thus, given time, the weights of the Boltzmann machine will stabilize so that the input state vector reaches an equilibrium.

Restricted Boltzmann machines (RBMs) are a special class of Boltzmann machines that are restricted in terms of the types of connections that are allowed. As with the case of Boltzmann machines, RBMs have two types of neurons: visible and hidden. Unlike Boltzmann machines which are fully-connected, RBMs do not allow for connections between two hidden neurons or connections between two visible neurons. The main advantage of these restrictions is that it allows for an easier implementation when compared to Boltzmann machines [41].

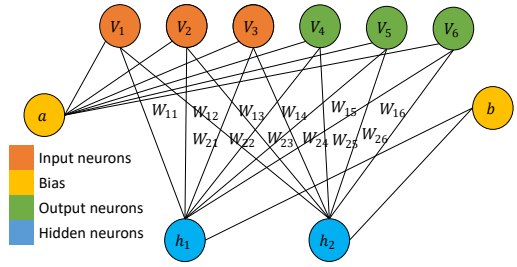


Fig. 14. Restricted Boltzmann machine with connection only between the hidden neurons and the visible neurons (input and output neurons).

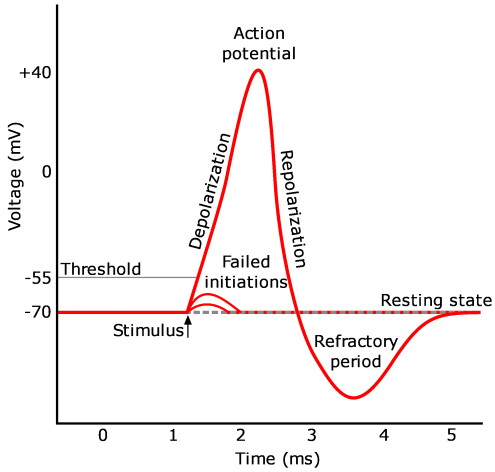


Fig. 15. Electrical response characteristic of the Hodgkin-Huxley model[3].

H. Biologically Plausible Neurons

Though artificial neural networks models described thus far have proven to be resilient and versatile, they are not very biologically plausible. These models fail to account that spikes are the tokens of information processing in the brain. Furthermore, they do not account for that the strength of communication is encoded in the synapses. One of the most important works was the development of a conductance-based model of the neuron by Hodgkin and Huxley in 1952 [42]. This was a mathematical model describing the electrical characteristics of neurons through a set of nonlinear differential equations. They received the 1963 Nobel Prize in Physiology or Medicine for their work describing the initiation and propagation of action potentials.

The electrical characteristics of the Hodgkin-Huxley model are shown in Fig. 15. Every neuron has a resting state which defines the membrane potential (MP) of the neuron in the absence of stimuli. An external stimulus to the neuron will increase its MP by a certain amount. When this increase is insufficient to make the MP exceed the threshold it is referred to as a failed initialization and the MP decays back to the

[3]https://commons.wikimedia.org/w/index.php?curid=2241513

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/OJCAS.2020.3032092, IEEE Open Journal of Circuits and Systems

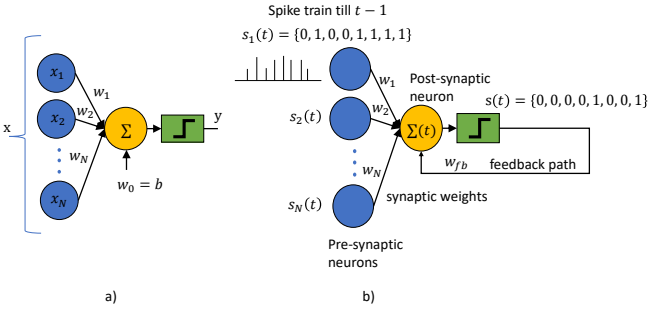


Fig. 16. Differences between an artificial neural network and a spiking neural network.

resting state. When the stimulus causes the MP to exceed the threshold, there is a rapid depolarization that cause an increase in the voltage. This is referred to as the action potential and can be interpreted as the firing of the neuron. Post the action potential the neurons re-polarize back to a negative potential with a refractory period that actively suppresses further firing due to external stimuli.

I. Hyperdimensional Computing

Hyperdimensional (HD) computing is a new computing paradigm based on the cognitive model [43] to define the binding problem of connectionist models. HD computing relies on the high dimensionality, randomness, and the abundance of nearly orthogonal vectors [44]. Traditional computing approaches rely on bits to encode data, and all operations are deterministic and require much hardware compute resources. However, HD computing uses a hypervector representation where the dimensionality is in the order of thousands. These ultra-wide words introduce redundancy against noise, and are, therefore, inherently robust [45]. HD computing supports three types of primary data transformation operations. Firstly, addition operation or *bundling* of hypervectors involves the point-wise addition of all hypervectors, and the result is binarized based on a majority rule threshold. Secondly, the multiplication operation or *bundling* forms associations between two related hypervectors with the bitwise XOR function. The result of this operation is a new hypervector that is nearly orthogonal to both input vectors. Finally, the unary permutation operation shuffles the hypervector resulting in a new permuted hypervector that is quasi-orthogonal to the original input. A detailed review of HD computing can be found in [44].

III. Introduction of Deep Neural Network Models

A. Convolutional Neural Networks (CNNs)

In the NeoCognitron model, all the S cells of a plane would have an identical response to a fixed input. This is mathematically equivalent to performing a convolution operation with a fixed filter across the input image. Similarly, the operation of the C cells is very similar to a ReLU followed by a pooling operation. Also, NeoCognitron and other early models were primarily unsupervised; however, these could be

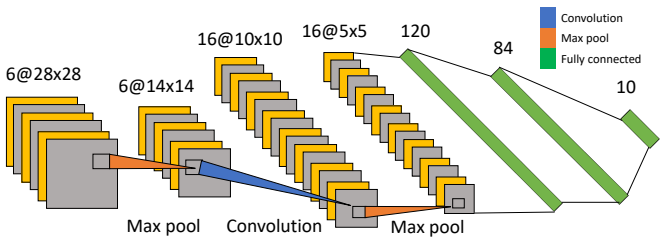


Fig. 17. Structure of LeNet-5 [47].

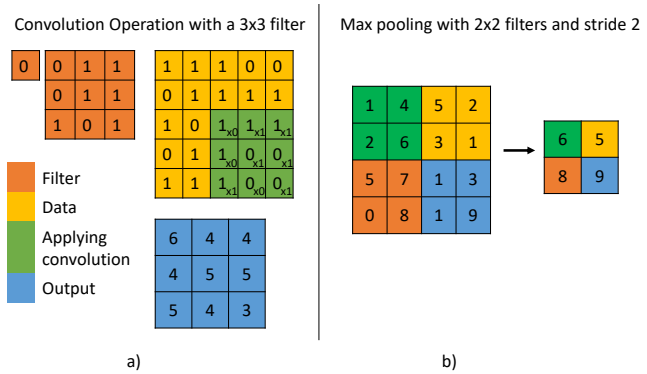


Fig. 18. Fundamental layers in a CNN. a) Convolutional layer. b) Pooling layer.

efficiently converted into a supervised problem with the insertion of additional decision-making neurons for the different classes. These simple extensions allow for the use of external supervision and backpropagation to train the neural network for a classification task. This was the principle behind the development of one of the first convolutional neural networks (CNNs) for the handwritten digit recognition task [46]. The development of the LeNet-5 model serves as a baseline for most modern CNN applications [47].

A convolutional neural network comprises of *convolutional* and *downsampling* or pooling layers. These are usually in sequence and can occur in any order. These layers are usually followed by one or more fully-connected (FC) layers or linear layers to perform the classification or regression task.

The essential operation of a convolutional layer is shown in Fig. 18(a), where the operands are the input data and the filter. The convolutional layer is a misnomer as it performs a correlation operation between the filter and the input and is summarized by Eq. (8).

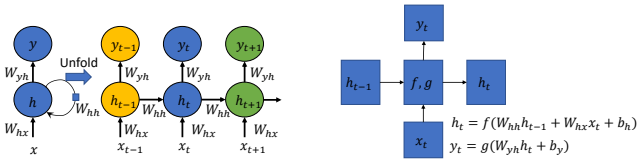
$$Y(x,y) = \sigma\Big(\sum_{i=0}^{k-1}\sum_{j=0}^{k-1}\sum_{c=0}^{C-1} W(i,j,c)X(x+i,y+j,c)+b\Big) \quad (8)$$

where W represents the filter of size $k \times k \times C$, b is the bias term, X is the input data, C represents the number of inputs channels and is also referred to as feature maps, and $Y(x,y)$ represents the output of the convolution operation at location x and y. Here σ represents a non-linear activation function such as sigmoid, tanh, or ReLU. The convolution layer scans the regions of the two-dimensional space, multiplying the filters' coefficients with each region's data. After the convolution

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/OJCAS.2020.3032092, IEEE Open Journal of Circuits and Systems

operation is completed, the resultant output is passed through a nonlinear activation layer.

A convolution operation often creates much redundancy on the output; therefore, it is beneficial to subsample the output by aggregating the information of a region into a single pixel. A pooling or downsampling layer is designed to achieve that goal and can be interpreted as scanning the input feature and performing a fixed operation on a localized region, most commonly the max or average operation. This layer performs subsampling as the stride of the operation is of the same order as its region size. Its function is to progressively reduce the spatial size of the representation to reduce the number of parameters and computation in the network. For the convolutional layers and any MLP layers, the filter or connection parameters have to be learned from the training data for the targeted classification or regression task. However, pooling layers are often fixed and cannot be learned.



(a) (b)

Fig. 19. Direct implementation of recurrent neural network (RNN): a) Unrolling a RNN in time. b) Single RNN unit operation.

B. Recurrent Neural Networks (RNNs)

Modern applications such as speech recognition and language translation require machine learning models to learn and predict patterns from signals, text and images. The input data x consists of a sequence of correlated data points $\boldsymbol{x}_t$. A recurrent neural network (RNN) is a generalization of a feedforward neural network with internal memory stored in the form of learnable weights [48], [49] . RNN inherently computes output $\boldsymbol{y}_t$ from the input datapoint $\boldsymbol{x}_t$ and its past output. Unlike feedforward neural networks, RNNs can use their internal state (memory) to process sequences of inputs. The ability of the RNN to *remember* information through its state makes it extremely suitable for sequence-to-sequence translation tasks such as speech recognition and language translation. Fig. 19(a) shows the typical structure of an RNN unfolded in time with input sequence $\boldsymbol{x}_t$ and output sequence $\boldsymbol{y}_t$. The hidden state, $\boldsymbol{h}_t$, is the *memory* of the network and is given by Eq. (9) and the output, $\boldsymbol{y}_t$, is defined by Eq. (10).

$$\boldsymbol{h}_t = f(\boldsymbol{W}_{hx}\boldsymbol{x}_t + \boldsymbol{W}_{hh}\boldsymbol{h}_{t-1} + \boldsymbol{b}_h) \tag{9}$$

$$\boldsymbol{y}_t = g(\boldsymbol{W}_{yh}\boldsymbol{h}_t + \boldsymbol{b}_y) \tag{10}$$

where f and g are non-linear functions such as tanh, sigmoid or ReLU, $\boldsymbol{W}_{hh}$, $\boldsymbol{W}_{hx}$, $\boldsymbol{W}_{yh}$, $\boldsymbol{b}_y$ and $\boldsymbol{b}_h$ represent the learnable weights and biases for the network. The computation of the hidden state $\boldsymbol{h}_t$ and output sequence $\boldsymbol{y}_t$ is depicted in Fig. 19(b). A popular variant of RNNs is the *bidirectional* RNN [50].

RNNs have two significant drawbacks. First, as the gradient is back-propagated to earlier time step, this repeated propagation may significantly reduce the scale of the gradients. As the time depth increases, the gradient *vanishes*, and the parameters stop learning. As a result of this vanishing gradient problem, as the network goes further back, its performance gets saturated. Second, RNNs are good at understanding the short-term dependencies in the data; however, they suffer considerably trying to capture long-term dependencies [51], [52].

C. Spiking Neural Networks

Spiking neural networks (SNNs) [53] bridge the gap between ANNs and more biologically plausible models. The differences between ANNs and SNNs are highlighted in Fig. 16. ANNs operate on real numbers and process all the inputs at a time. SNNs, however, work on time-dependent data with the information encoded into the spike train. The *membrane potential* of a neuron is modeled to define the amount of electric charge stored in the neuron. The membrane potential accumulates the charge from the pre-synaptic neurons to determine whether the neuron should fire.

As the focus is on more biological models, there also needs to be a good understanding of how these neurons can learn. However, backpropagation, though an excellent optimization procedure, does not correlate to the brain and does not mimic actual learning. Revisiting Hebbian theory, an important consideration for synaptic strength is not just the synchronicity of neuron firing but also the temporal relationship between the firing of different neurons. Synaptic time-dependent plasticity (STDP) quantifies the strengthening or weakening of the conductivity of the synapse based on the relative order of firing between the pre-synaptic neurons and post-synaptic neurons shown in Fig. 16(b).

The characteristics of STDP for an excitatory to excitatory neuron connection [54] are shown in Fig. 20. Δt defines the relative time between the firing of the the post and pre-synaptic neurons ($\Delta t = t_{\text{post}} - t_{\text{pre}}$). Simultaneous firing of the neurons, where the pre-synaptic neuron precedes the post-synaptic neuron ($\Delta t > 0$), called a causal firing, leads to the long term potentiation (LTP) of the neurons. This causal relationship between input and output increases the synapse's conductance and furthers the connection between the two neurons. An anti-causal relationship is established when the pre-synaptic firing succeeds the post-synaptic firing ($\Delta t < 0$). This relationship causes a decrease in synapse's conductivity and leads to long-term depression (LTD) of the neurons. For either of the effects, the switching must occur in close proximity as the effects decrease with an increase in the time difference [55].

One of the simplest models to simulate the membrane potential is the integrate and fire model [56], [53]. The model can be assumed to have infinite memory where the membrane potential contains the weighted history of all inputs as shown in Eq. (11)

$$u_j(t) = \sum_{i \in \mathcal{P}_i} \sum_{\tau=1}^{t} w_{ij} s_{i,\tau} \tag{11}$$

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/OJCAS.2020.3032092, IEEE Open Journal of Circuits and Systems

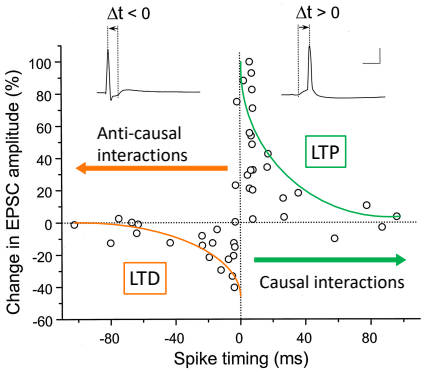


Fig. 20. Excitatory postsynaptic potential change with the causality of the presyapctic stimulus highlighting the synaptic time dependent plasticity, long term potentiation and depression, behavior of neurons. Adapted from [55].

where τ is the time step upto the current time t in the spike train of neuron $s_{i,\tau}$. $s_{i,\tau}$ denotes the input neuron $\mathcal{P}_i$ from the pre-synaptic set $\mathcal{P}$ at time τ. $u_j(t)$ is the membrane potential of post-synaptic neuron j and w_{ij} is the weight of the synapse from neuron i to neuron j. The neuron j will fire if the membrane potential exceeds a certain threshold $u_j(t) > \theta_j$, where θ_j is the threshold of neuron j. The dynamic behaviour of this model is shown in Fig. 21(a).

This simple model caters well to approximate the basic characteristic of the neuron; however, it fails to capture the time dependency on the output. In particular, the likelihood of a post-synaptic neuron to fire decreases with time from the incident pre-synaptic fire. This general forgetfulness of the neurons [53] is captured in the membrane potential as shown in Eq. (12) representing a *spike response model* (SRM)[57], [58] in the discreet time-space. The leaky integrate and fire (LIF) model is a special case of the SRM without time-dependent weight functions [57].

$$u_j(t) = \sum_{i \in \mathcal{P}_\rangle} \sum_{k=1}^{(K)_i} w_{ij,k}(\alpha(t) * s_{i,t})(k) + w_j(\beta(t) * s_{j,t})(k) + b_j \tag{12}$$

where $w_{ij,k}$ represents the time-dependent kernel weight function that has trainable weights up to $(K)_i$ time units, $\alpha(t)$ and $\beta(t)$ are the exponential decay functions for the feedforward and feedback paths, respectively, b_j represents the bias of the function and $*$ represents the convolution operation. The dynamic behavior of this model is shown in Fig. 21(b).

SNNs have higher information representational capacity due to their temporal dimension, and it is believed that this gives them an advantage over the traditional methods [60]. Thus, SNNs are well suited for processing spatio-temporal event-based information. This makes them successful in fields like event based cameras and vision [61]. As these neurons' activations are sparse and event-driven, SNNs can be more energy efficient compared to CNNs. Furthermore, due to the their event-driven nature, the latency of computation in SNNs is less. In an SNN, although the accuracy of the output is degraded initially, the accuracy improves as more inputs are processed. Even when more data are not available, the accuracy can be improved by retraining the weights [62].

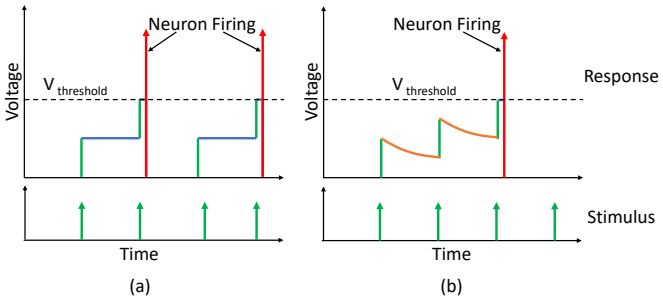


Fig. 21. Response characteristic of SNN models. a) Integrate and fire model. b) Leaky integrate and fire model. [59]

In terms of training, STDP has served as a basis for many training algorithms [63], [64]. These algorithms are trained based on the relative timing of spikes. However, there has also been research into non Hebbian-based leaning [65]. Other training algorithms also exist based on evolutionary techniques [66]. Another general approach is to train traditional networks using backpropagation and then convert these networks to SNNs to utilize them as efficient inference engines [67]. Finally, training algorithms can use approximations of the gradient descent by formulating the training task as an optimization problem [68] that can be used to train spiking-based CNN architectures [60].

The major shortcoming with SNNs is that they typically do not attain the same accuracy as traditional neural network approaches. While there are numerous large training datasets and benchmarks available for CNNs, very few benchmarks are available for SNNs. Accuracy of SNNs could improve with increase in the data available for applications that can benefit from these networks. Furthermore, a careful trade-off analysis is needed between their advantages versus the accuracy of the output. Furthermore, to best utilize the advantages of SNNs, it should be run on neuromorphic hardware creating a barrier to entry due to lack of availability, compared to traditional approaches.

IV. Advanced Deep Neural Networks

A. Image Classification

The re-emergence and popularity in deep learning have, in large part, to do with its success in the image classification task. A lot of this credit goes to two major driving points: the availability of large data sets and the availability and adoption of new computing resources. In particular, image classification was a very daunting task that had only received marginal gains with time. To overcome these shortcomings, Li, a computer science professor, curated and created a large dataset of labeled images ImageNet [10]. This tackled the fundamental limitation of existing algorithm-based approaches where algorithms would not work as well if the data were not representative of the real world. Thus ImageNet acted as a bridge to classify and categorize a sizable hierarchical database. The ImageNet database also evolved into a competition for image classification, the ImageNet Large-Scale Visual

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/OJCAS.2020.3032092, IEEE Open Journal of Circuits and Systems

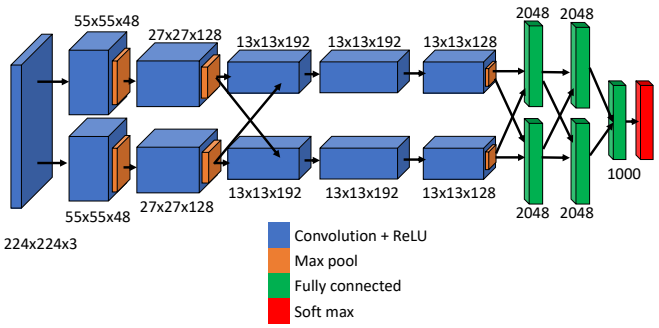


Fig. 22. Neural network architecture of AlexNet [69].

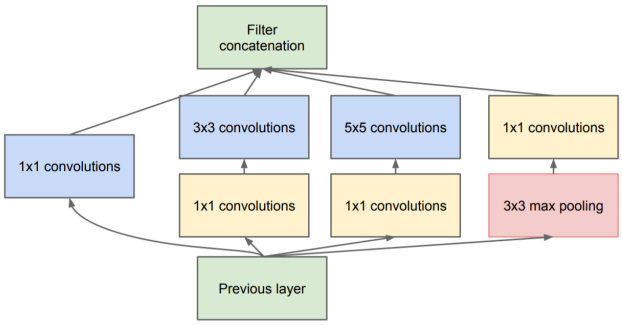


Fig. 23. Neural network architecture of an inception module in GoogLeNet. Taken from [70].

Recognition Challenge (ILSVRC). The goal of the completion was for researchers to obtain the lowest top-1 and top-5 error rates. This catalyzed the boom in interest and research into AI as it gave a stable foundation for developing more advanced models and approaches.

1) AlexNet

The AlexNet [69] neural network architecture emerged as part of the ILSVRC classification task. With the abundance of data in the training set, it was a challenge to design and develop a model that would be large enough to tackle the task at hand and fast enough to peruse the dataset. It was one of the first *deep* neural networks with five convolutional layers and three fully-connected layers. The model consisted of more than 60 million parameters and 650,000 neurons, taking five to six days to train on two graphics processing units (GPUs). It is important to point out that the convolutional layers in AlexNet contribute to about 6% of the parameters but 95% of the computations. Thus, the convolutional layers are computationally intensive while the fully-connected layers are memory-intensive.

Some of the characteristics that made AlexNet viable were the use of non-saturating units, ReLU [71], reducing overfitting with dropout, and the use of GPUs to accelerate the training process. The new activation unit ReLU can reach a 25% training error rate six times faster on the CIFAR-10 [72] dataset over conventional tanh activation. A traditional approach to overfitting involved regularization and the use of multiple

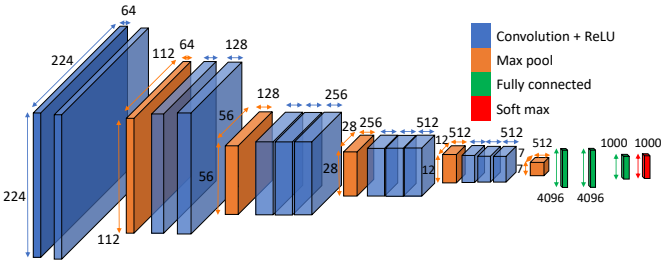


Fig. 24. Neural network architecture of VGG-16 [77].

models to predict the outcome. However, with significant run times for training large models, dropout [73] was the preferred approach. Dropout determines the probability that a given neuron should include or *dropout* a neuron for that training input. This ensures that neurons do not become overly reliant on other neurons and learn more robust features. Finally, It was one of the early papers to use GPUs [74], [75], [76], and to adopt the use of multiple GPUs to fit larger models that would not fit in a single GPU.

The model stood out at its time, winning the 2012 ILSVRC with a top-5 accuracy of 15.3% with the nearest competitor at 26.2%, cementing CNNs as the premier model for image classification.

2) VGG

VGG [77] was a neural network architecture that was the runner-up in the 2014 ILSVRC classification task. The innovation in the approach was using smaller convolutional filters that significantly increased the network's depth. Fig. 24 shows the network corresponding to architecture D of the paper, with 16 layers. There are 13 convolutional layers, where each layer uses a small 3×3 filter with a stride of 1. There are five max-pooling layers, performing a spatial pooling over a 2×2 pixel window, with stride 2. Finally, there are three fully-connected layers, the first two with 4096 channels each, and the third with a 1000 outputs for the ILSVRC classification task. This 16 layer design has close to 140 million parameters, and it was able to achieve a 6.8% top-5 error rate.

3) GoogLeNet

GoogLeNet [70], the winning entry in the 2014 ILSVRC classification task, was a very deep network with 22 layers. The model's essential premise was that the salient features of an image could have extreme size variations. Therefore, deciding an appropriate kernel size for the model is a challenging task. A large kernel is useful to capture more global attributes, whereas a small kernel is ideal for picking up local attributes of the image. To overcome these issues, the network makes use of *inception* modules. An inception mode primarily contains multiple filters of different sizes $(1 \times 1, 3 \times 3, 5 \times 5)$ that perform their computations at the same level. Additional 1×1 filters are used to perform dimensionality reduction to reduce the computational complexity of the network.

Fig. 23 shows the structure of an inception module in the inception-v1 architecture of GoogLeNet. GoogLeNet consists of three initial convolutional layers that work on the input image, followed by nine inception modules stacked linearly. Finally, it has a single, fully-connected layer to perform the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/OJCAS.2020.3032092, IEEE Open Journal of Circuits and Systems

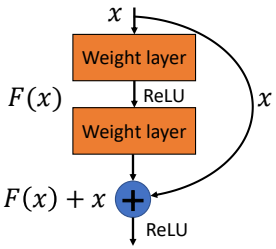


Fig. 25. Architecture of the residual layer in ResNet [2].

classification task. As it is an extremely deep classifier, it has issues with vanishing gradients. Thus, auxiliary classifiers were introduced at intermediate layers during the training phase. These auxiliary classifiers contribute to the loss of the systems, and the total loss function is a weighted sum of the auxiliary loss and the real loss. With approximately 6.8 million parameters, $12\times$ fewer parameters than AlexNet, it achieves a top-5 error rate of 6.8%.

4) ResNet

Conventional wisdom states that additional layers in the model should increase its accuracy if the problem is complicated enough. However, it is seen that the deeper the networks get, the more they suffer from the vanishing gradient problems. Beyond a certain depth, the model training suffers and does not recover. Prior approaches to solving this problem included using auxiliary classifiers that were introduced at intermediate layers, like the ones used in GoogleNet. For extremely deep networks (>30 layers), the authors of [2] hypothesize that it is easier for neural network layers to learn a residual mapping function, $F(X) + X$, than a direct implementation. The residual layer structure is shown in Fig. 25, where there are *shortcut* connections across layers and an element-wise addition. This can be taken as a special case of the highway networks [78], [79], unrolled feedforward networks based on LSTMs, where the input, X, is multiplied by an identity matrix rather than an explicit gating parameter to be learned. These differences significantly reduce the number of parameters to be learned, and it is shown that this simpler residual mapping helps achieve the state of art results. The innovations in these residual nets or ResNets enabled these models to rank first in the 2015 ILSVRC and COCO [11] competition in the detection, localization, and segmentation tasks.

B. Image Segmentation

1) Fully convolutional networks

Unlike a classical image classification problem where the output is a single label, image segmentation requires each pixel in the input image to be assigned to a specific class. The authors of [80] were the first to propose a fully convolutional network for this application. The network consists purely of convolutional and max pooling layers without any fully connected layers. The fully convolutional nature provides two advantages: (i) reusability of the model on an image of any size since the trained model only stores convolution filter weights and (ii) training and inference speedup compared to networks with fully-connected layers. For example, as stated in [80], although AlexNet takes 1.2 ms (on a typical GPU) to produce the classification scores of a 227 $\times$ 227 image, the fully convolutional network takes 22 ms to produce a 10 $\times$ 10 grid of outputs from a 500 $\times$ 500 image, which is more than five times faster than the naive approach.

The main idea in [80] is to cast the fully-connected layer in the LeNet into a convolutional layer with a kernel that covers the entire image. This converts the architecture into a fully-convolutional network. However, due to the subsampling nature convolution and max pooling, it results in a reduced dimensionality output image when compared to the input. For an image segmentation task, the input and output image dimensions are identical since each pixel must be classified. The lost dimensionality is recovered using deconvolutions/upsampling (similar to the decoder networks in autoencoders). The deconvolution filter weights are learned during training, similar to the convolution filter weights, creating an end-to-end fully convolutional structure for image segmentation.

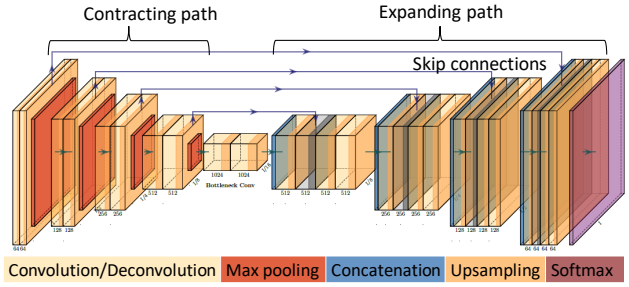


Fig. 26. Architecture of U-Net [81].

2) U-Net

A popular extension of the architecture proposed in [80] is U-Net [81], which is very successful in biomedical image segmentation applications. The architecture of this network is shown in Fig. 26. At the highest level, it is a fully convolutional network comprised of two parts: (i) a contracting path and (ii) a symmetric expanding path. Also, it consists of *skip connections* [82], which concatenates higher-level features from the contracting path into the expanding paths, which enable faster convergence during training. An essential modification from the work in [80] is that the U-Net has a large number of feature channels during upsampling. This allows the network to propagate context information to higher resolution layers. U-Net architecture's fully-convolutional nature allows for easy segmentation of large-sized input images, which would otherwise be limited by large dimensions of the fully-connected layer. This U-Net architecture requires fewer data points for training and results in more precise image segments than the work in [80].

C. Language and Time-Series Processing

1) Long Short-Term Memory (LSTM)

Long short-term memory (LSTM) networks are special types of networks capable of learning long-term dependencies

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/OJCAS.2020.3032092, IEEE Open Journal of Circuits and Systems

in time-series data. It was first introduced in [83] by Hochreiter and Schmidhuber (1997), and further expanded in [84], [85], [86]. However, these models did not gain traction until after Graves's Ph.D. Thesis [87].

Unlike the RNNs, which consist of a single neural network, the LSTM cell consists of four neural networks interacting uniquely. A standard LSTM cell is shown in Fig. 27 with the four neural networks. The core idea is that the cell state, C_t, is changed with minor linear interactions. The cell consists of three regulator structures called *gates* responsible for altering the cell state. Each gate consists of a sigmoid layer that outputs a number between zero and one, describing how much of each component should pass through the gate (zero is nothing, and one is everything). The three gates are referred to as *forget gate*, *input gate*, and *output gate*.

The forget gate is responsible for deciding what information to let through to the cell state. The input gate has two layers: a sigmoid layer and the tanh layer.

Together, the layers decide which values of the current state to update and generate a vector of new candidate values to be added to the current state. The equations representing the operation of the LSTM cell are given in Eqs. (13) to (18). Eq. (13) represents the forget gate operation. Eqs. (14) to (16) together represent the input gate and update step, and Eqs. (17) and (18) represent the operation of the output gate.

$$\boldsymbol{f}_t = \sigma(\boldsymbol{W}_f[\boldsymbol{h}_{t-1}, \boldsymbol{x_t}] + \boldsymbol{b}_f) \tag{13}$$

$$\boldsymbol{i}_t = \sigma(\boldsymbol{W}_i[\boldsymbol{h}_{t-1}, \boldsymbol{x}_t] + b_i) \tag{14}$$

$$\widetilde{\boldsymbol{C}}_t = \tanh(\boldsymbol{W}_c[\boldsymbol{h}_{t-1}, \boldsymbol{x}_t] + \boldsymbol{b}_c) \tag{15}$$

$$\boldsymbol{C}_t = \boldsymbol{f}_t \boldsymbol{C}_{t-1} + \boldsymbol{i}_t \widetilde{\boldsymbol{C}}_t \tag{16}$$

$$\boldsymbol{o}_t = \sigma(\boldsymbol{W}_i[\boldsymbol{h}_{t-1}, \boldsymbol{x}_t] + \boldsymbol{b}_o) \tag{17}$$

$$\boldsymbol{h}_t = \boldsymbol{o}_t \tanh(\boldsymbol{C}_t) \tag{18}$$

where $\boldsymbol{i}_t$ is the output mask of the input gate, $\boldsymbol{o}_t$ is the output mask of the output gate, $\boldsymbol{f}_t$ is the output mask of the forget gate, $\boldsymbol{h}_t$ is the output of the LSTM block, and $\boldsymbol{x}_t$ is the input data.

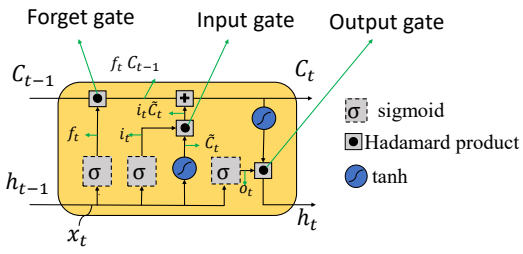


Fig. 27. Implementation of an LSTM module.

2) Gated Recurrent Unit

The most commonly used recurrent network model has been LSTM, which can learn long-term dependencies. We focus on a popular LSTM variant, referred to as gated recurrent unit (GRU). A gated recurrent unit (Fig. 28(b)) (GRU) was proposed in [88] where each recurrent unit can adaptively capture dependencies of different time scales. The GRU is similar to the LSTM cell with gates modulating the flow of information through the cell. Fig. 28 highlights the differences between a standard LSTM cell and a GRU cell where Fig. 28(a) is a high-level representation of the LSTM cell shown in Fig. 27. This higher-level representation helps us understand the difference between the GRU clearly. The main differences between the GRU cells and LSTM cells are listed below:

1) In the LSTM unit, the output gate controls the amount of the memory content that is seen or used by other units in the network. On the other hand, the GRU exposes its full content without any control.
2) The location of the input gate or the corresponding reset gate is different between the two.
3) The LSTM unit computes the new memory content without any separate control of the amount of information flowing from the previous time step. Rather, the LSTM unit controls the amount of the new memory content being added to the memory cell independently from the forget gate. On the other hand, the GRU controls the information flow from the previous activation when computing the new candidate activation but does not independently control the amount of the candidate activation being added (the control is tied via the update gate).

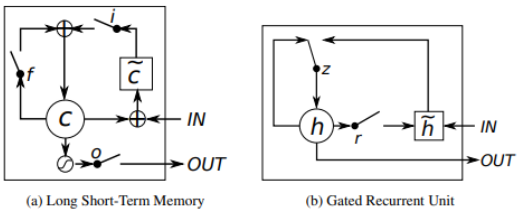


Fig. 28. Differences between an LSTM cell and a GRU cell [88].

From these similarities and differences alone, it is difficult to conclude which types of gating units would perform better in general. Although [88] reported that these two units performed comparably to each other according to their experiments on certain tasks, it is unclear whether this applies as well to tasks other than machine translation.

D. Latent Space Generative Models

Most common forms of generating a latent space generative models are derived from a class of learning methodologies referred to as autoencoders. Autoencoders [89] represent a form of unsupervised learning where the goal is to predict the input from itself. To ensure that the network does not directly represent the input, autoencoders often have less complexity than the input space to force the network to learn a more optimized representation of the underlying data [90]. This serves two purposes. First, it can perform a lossy compression of the data with the *encoder* and reconstruct the data from the compressed form with the *decoder*. Second, by reducing dimensionality during encoding, the autoencoder learns the data's underlying characteristics and thereby creates a new feature space that defines and captures the relationships between inputs for a given task. Some of the common variants

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/OJCAS.2020.3032092, IEEE Open Journal of Circuits and Systems

of autoencoders include sparse autoencoders [91], variational autoencoders [92], and denoising autoencoders [93]. Autoencoders can potentially lead to non-conventional solutions to conventional problems in communications such as coding for data transmission [94].

E. Graph-Based Data Processing

Graph Neural Networks (GNNs) [95], [96] are similar to RNNs but can be applied to more general class of graphs. GNNs can operate on graphs that cannot be represented in a grid-like structure and are appropriate for application domains such as social networks and telecommunication networks. The ultimate goal of a GNN is to derive an embedding matrix that can be used to transform the input feature space into the desired output label or attribute. The input features of the graph contain the features of the node itself and its edges as well as some function of the information about its neighborhood as shown in Fig. 29. The operation on the GNN can be summarized by Eqs. (19) to (22) and [96], [97]:

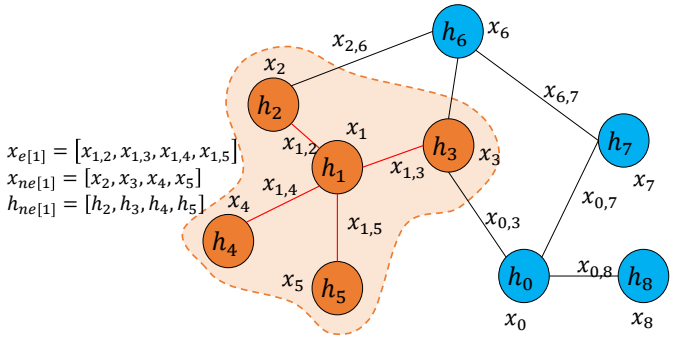


Fig. 29. Graph neural networks, adapted from [96]. Features of node x_1 dependence on its neighborhood.

$$\boldsymbol{h}_v = f(\boldsymbol{x}_v, \boldsymbol{x}_{e[v]}, \boldsymbol{h}_{ne[v]}, \boldsymbol{x}_{ne[v]}) \tag{19}$$

$$\boldsymbol{o}_v = g(\boldsymbol{h}_v, \boldsymbol{x}_v) \tag{20}$$

$$\boldsymbol{H}^l = F(\boldsymbol{H}^{l-1}, \boldsymbol{X}) \tag{21}$$

$$\boldsymbol{O} = G(\boldsymbol{H}, \boldsymbol{X}_n) \tag{22}$$

Eq. (19) defines the shared local transition function $f(.)$ that processes the information from a node and its neighbors to generate a D-dimensional state vector $\boldsymbol{h}_v$ of the node v, where $\boldsymbol{x}_v$ denotes the features of the node; $\boldsymbol{x}_{e[v]}$ are the features of the edges of the node, $e[v]$; $\boldsymbol{h}_{ne[v]}$ represents the state vector of the nodes in the neighborhood of the node, $ne[v]$; and $\boldsymbol{x}_{ne[v]}$ represents the features of the nodes in the neighborhood of the node. Eq. (20) defines the shared local output function $g(.)$ that processes the current node features, $\boldsymbol{x}_v$, and state vector, $\boldsymbol{h}_v$, to generate an output vector $\boldsymbol{o}_v$ that represents the label or value we are interested in computing. The computations described in $f(.)$ and $g(.)$ are equivalent to feedforward neural networks. The training of GNNs involves training the weights of these feedforward neural networks.

A compact form of Eq. (19) can be represented in terms of iterative function of matrices as shown in Eq. (21), where $\boldsymbol{H}^l$ represents the embedding matrix or state matrix by stacking embedding vectors $\boldsymbol{h}_v$ at iteration l. F represents the global transition function that performs $f(.)$ node-wise, and $\boldsymbol{X}$ is the set of all input features of all nodes and edges. Similarly, Eq. (22) represents a compact form of Eq. (20) where G is the global output function obtained from evaluating g node-wise and $\boldsymbol{X}_n$ is the set of all input features of all nodes.

1) Graph Convolution Networks

Graph Convolution Networks (GCNs) can be interpreted as extensions of conventional CNNs into the graph domain. There are two general classes of approaches to this problem: spectral and non-spectral. In the spectral approach [98], the convolution operation is defined in the Fourier domain to optimize the number of computations. This is further enhanced by introducing the spectral filters with smooth coefficients [99]. Later enhancements include the approximation of the filters employing a Chebyshev expansion of the graph Laplacian [100].

GCNs are a general class of networks that try to simplify the existing spectral methods. Every node, v, is characterized by a feature vector $\boldsymbol{x}_v$, that when stacked together, form the feature matrix $\boldsymbol{X}$. The network also processes information about the graph's structure, usually in the form of an adjacency matrix $\boldsymbol{A}$. The result is an output matrix, $\boldsymbol{Z}$, that contains the output labels or features. A layer in the network can be represented in a general form as shown in Eq. (23), where $\boldsymbol{H}^l$ represents the feature matrix at layer l, $\boldsymbol{H}^0 = \boldsymbol{X}$ and $\boldsymbol{Z} = \boldsymbol{H}^L$. A simplified approach to the problem is to design filters that only look at the immediate neighborhood of the current node [101]. With this, a graph convolutional layer can be described by Eq. (24).

$$\boldsymbol{H}^l = F(\boldsymbol{H}^{l-1}, \boldsymbol{A}) \tag{23}$$

$$F(\boldsymbol{H}, \boldsymbol{A}) = \sigma(\hat{\boldsymbol{D}}^{-\frac{1}{2}}\hat{\boldsymbol{A}}\hat{\boldsymbol{D}}^{-\frac{1}{2}}\boldsymbol{H}\boldsymbol{W}) \tag{24}$$

where $\hat{\boldsymbol{A}} = \boldsymbol{A} + \boldsymbol{I}$ is the adjacency matrix with an addition of the identity matrix to create self-loops to account for the current node being processed, $\hat{\boldsymbol{D}}$ represents the degree matrix of the node and is used to normalize the adjacency matrix. $\boldsymbol{W}$ is the weight matrix of the layer. This layer structure is analogous to a spatial convolution layer, as multiplying the adjacency matrix with the state matrix aggregates the information from the neighboring nodes into the central node.

These examples have focused on a simplified spectral approach; however, there are multiple non-spectral approaches [102], [103], [104] that define convolutions directly on the graph, operating on spatially close neighbors. Different techniques introduced include a specific weight matrix for each node degree [102], weights for each input channel, and neighborhood degree [103], normalizing neighborhoods containing a fixed number of nodes [105]. Other notable approaches include [106], [107]. A review of these approaches can be found in [108].

2) Graph Attention Networks

Graph attention networks (GATs) were introduced in 2018 [109] as an attention-based architecture to perform node classification of graph-structured data. This computes each node's hidden representations by attending over its neighbors, following a self-attention strategy. This has advantages over

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/OJCAS.2020.3032092, IEEE Open Journal of Circuits and Systems

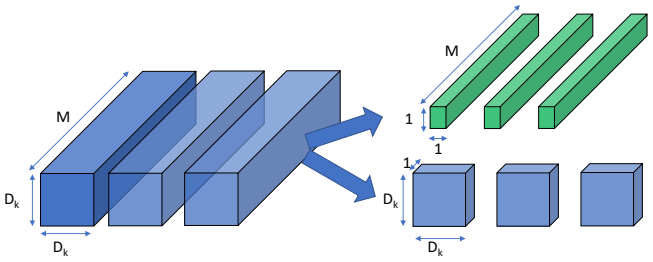


Fig. 30. MobileNet-v1 architecture, taken from [110]. A standard convolution converted to a separable convolution operation. The new stack is highlighted on the right

the GCNs. First, it is highly computationally efficient as the self-attention layer's operation is parallelized. Second, unlike GCNs, it allows assigning different degrees to nodes of the same neighborhood. Lastly, it is directly applicable to inductive learning.

F. Resource-Constrained Models

There has been a significant interest in developing networks designed explicitly for mapping to resource-constrained systems like microcontrollers. Examples of this type of network include MobileNet [110] and its variants like ShuffleNet [111]. MobileNet adapts the concept of bottleneck layers introduced in GoogleNet to minimize the total number of computations in the network. It consists of a set of separable convolution kernels that are depthwise convolution operations for spatial domain and pointwise convolution operations in the channel domain. This is illustrated in Fig. 30. MobileNet takes this a step further by allowing a further trade-off between the accuracy of the model and the neural network size to create models that can fit into various devices. Also pruning techniques, such as the lottery ticket hypothesis [112], have enabled high accuracies with very few parameters.

V. Conclusion and Summary

Machine leaning and AI have a come a long way since its early days and brain-inspired computing has driven a lot of these innovations. This evolution over decades has led to each generation of models tackling the problems of the last generation and improving on them. With vast explosion in the amount and types of data collected, these brain inspired models have had and will continue to have major impacts in the foreseeable future. The Association for Computing Machinery (ACM) awarded the 2018 Turing Award to Bengio, Hinton and LeCun for "conceptual and engineering breakthroughs that have made deep neural networks a critical component of computing."

This paper has presented an overview of the evolution of these brain-inspired models with a historical context. The paper presents a unique perspective with respect to the advantages and limitations of each model. A proper understanding of the limitations of each model is only complete by knowing the limitations of the environment in which they were developed.

While this paper overviews brain-inspired models, significant research efforts have been directed towards design of general-purpose energy-efficient hardware accelerators [113], [114], [115]; this topic is beyond the scope of this paper. There have also been significant strides in the development of hardware accelerators for SNNs [116], [117], [118], CNNs [119], [120], [121], GNNs [122], [123] and training accelerators [124], [125], [126]. A comprehensive survey of the topic can be found in [127], [121]. We also refer the reader to recent research on *attention networks* [128] used in image captioning applications, *transformers* [129] used in natural language processing and on *neural architecture search* [130] to design neural network configurations that reduce the complexity of the network.

Despite significant progress in SNNs, more research should be devoted to developing SNN models and biologically plausible neural networks that achieve comparable accuracy to CNNs. This would be possible if benchmarks with large amount of data can be created for domains where SNNs can perform similar to CNNs or for domains where CNNs are not applicable.

Modern neural networks with many millions of parameters have outperformed classical methods that are based on linear system theory. Unfortunately, the linear systems cannot model the nonlinearities in the data or non-stationarity of stochastic data. The neural networks achieve superior performance due to the use of nonlinear activation functions. However, what is the least amount of nonlinearity that can be incorporated into traditional linear approaches to achieve performance similar to modern CNNs remains an open question. While this is a hard problem, any progress in this direction will lead to neural networks that are not as deep and contain far fewer parameters.

Deep learning has undergone a renaissance in the last decade in a number of fields like image processing, language processing, and graph processing to name a few. Advanced CNNs have reached super human level accuracies in a number of vision tasks like object detection, segmentation and classification. Recurrent networks and time based models have greatly advanced the state of the art in speech and language processing. Also, in an ever connected world graph neural networks have advanced our knowledge and processing of how different systems interact and how to interpret them.

The modern deep learning renaissance owes a lot to these pioneering historical models, that predated the availability of the resources needed to realize their true potential. These models will be further expanded and refined in coming decades as they find their applications in domains such as robotics and smart cars, drones, medical diagnostics and healthcare, and security.

References

[1] K. Olmstead, "Nearly half of Americans use digital voice assistants, mostly on their smartphones," *Pew Research Center*, 2017.

[2] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE Computer Society, 2016, pp. 770–778.

[3] S. Xie, R. Girshick, P. Dollar, Z. Tu, and K. He, "Aggregated residual transformations for deep neural networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[4] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. Alemi, "Inception-v4, inception-resnet and the impact of residual connections on learning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2017.

[5] D. Silver *et al.*, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[6] O. Vinyals *et al.*, "Grandmaster level in StarCraft II using multi-agent reinforcement learning," *Nature*, vol. 575, no. 7782, pp. 350–354, 2019.

[7] V. Gulshan *et al.*, "Development and validation of a deep learning algorithm for detection of diabetic retinopathy in retinal fundus photographs," *Journal of the American Medical Association*, vol. 316, no. 22, pp. 2402–2410, 12 2016.

[8] X. Jia and M. Q. Meng, "A deep convolutional neural network for bleeding detection in wireless capsule endoscopy images," in *Proceedings of the International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, 2016, pp. 639–642.

[9] G. Litjens, F. Ciompi, J. M. Wolterink, B. D. de Vos, T. Leiner, J. Teuwen, and I. Išgum, "State-of-the-art deep learning in cardiovascular image analysis," *Journal of the American College of Cardiology: Cardiovascular Imaging*, vol. 12, no. 8, Part 1, pp. 1549 – 1565, 2019.

[10] J. Deng, W. Dong, R. Socher, L. Li, Kai Li, and F. F. Li, "ImageNet: a large-scale hierarchical image database," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2009, pp. 248–255.

[11] T.-Y. Lin *et al.*, "Microsoft COCO: Common objects in context," 2014.

[12] M. Abadi *et al.*, "TensorFlow: A system for large-scale machine learning," in *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation*, 2016, pp. 265–283.

[13] A. Paszke *et al.*, "PyTorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems (NIPS)*, 2019, pp. 8026–8037.

[14] F. Van Veen, "The neural network zoo," Apr 2019. [Online]. Available: https://www.asimovinstitute.org/neural-network-zoo/

[15] T. Delbruck and S. Liu, "Data-driven neuromorphic DRAM-based CNN and RNN accelerators," in *Proceedings of the Asilomar Conference on Signals, Systems, and Computers*, 2019, pp. 500–506.

[16] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The Bulletin of Mathematical Biophysics*, vol. 5, no. 4, pp. 115–133, 1943.

[17] D. O. Hebb, *The Organization of behavior*. Wiley, New York, 1949.

[18] R. Morris, "D.O. Hebb: The organization of behavior, Wiley: New York; 1949," *Brain Research Bulletin*, vol. 50, no. 5, p. 437, 1999.

[19] F. Rosenblatt, "The perceptron: a probabilistic model for information storage and organization in the brain." *Psychological Review*, vol. 65, no. 6, p. 386, 1958.

[20] J. C. Hay, B. E. Lynch, and D. R. Smith, "Mark I perceptron operators' manual," Cornell Aeronautical Laboratory Inc. Buffalo, NY, Tech. Rep., 1960.

[21] Cornell Aeronautical Laboratory, "Mark I Perceptron," *Cornell University News Service records, #4-3-15. Division of Rare and Manuscript Collections, Cornell University Library*.

[22] M. Minsky and S. A. Papert, *Perceptrons: An Introduction to Computational Geometry*. The MIT Press, 2017.

[23] H. Block, "A review of ̈perceptrons: An introduction to computational geometry,̈" *Information and Control*, vol. 17, no. 5, pp. 501 – 522, 1970.

[24] M. Olazaran, "A sociological study of the official history of the perceptrons controversy," *Social Studies of Science*, vol. 26, no. 3, pp. 611–659, 1996.

[25] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Networks*, vol. 2, no. 5, pp. 359 – 366, 1989.

[26] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Mathematics of control, signals and systems*, vol. 2, pp. 303–314, 1989.

[27] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural Networks*, vol. 61, pp. 85 – 117, 2015.

[28] S. Dreyfus, "The numerical solution of variational problems," *Journal of Mathematical Analysis and Applications*, vol. 5, no. 1, pp. 30–45, 1962.

[29] S. Linnainmaa, "Taylor expansion of the accumulated rounding error," *BIT Numerical Mathematics*, vol. 16, no. 2, pp. 146–160, 1976.

[30] P. J. Werbos, "Applications of advances in nonlinear sensitivity analysis," in *System Modeling and Optimization*. Berlin, Heidelberg: Springer, 1982, pp. 762–770.

[31] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, *Learning Internal Representations by Error Propagation*. Cambridge, MA, USA: MIT Press, 1986, p. 318–362.

[32] D. H. Hubel and T. N. Wiesel, "Receptive fields of single neurones in the cat's striate cortex," *The Journal of physiology*, vol. 148, no. 3, p. 574, 1959.

[33] ——, "Receptive fields, binocular interaction and functional architecture in the cat's visual cortex," *The Journal of physiology*, vol. 160, no. 1, p. 106, 1962.

[34] ——, "Receptive fields and functional architecture of monkey striate cortex," *The Journal of Physiology*, vol. 195, no. 1, pp. 215–243, 1968.

[35] K. Fukushima, "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position," *Biological Cybernetics*, vol. 36, no. 4, pp. 193–202, 1980.

[36] A. Edalat. Neural networks and their applications. [Online]. Available: https://www.doc.ic.ac.uk/~ae/papers/Topics.pdf

[37] J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities," *Proceedings of the National Academy of Sciences*, vol. 79, no. 8, pp. 2554–2558, 1982.

[38] W. Little, "The existence of persistent states in the brain," *Mathematical Biosciences*, vol. 19, no. 1, pp. 101 – 120, 1974.

[39] G. Hinton and T. Sejnowski, "Learning and relearning in Boltzmann machines," *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, vol. 1, 01 1986.

[40] G. E. Hinton, "Boltzmann machine," *Scholarpedia*, vol. 2, no. 5, p. 1668, 2007, revision #91076.

[41] P. Smolensky, *Information Processing in Dynamical Systems: Foundations of Harmony Theory*. Cambridge, MA, USA: MIT Press, 1986, p. 194–281.

[42] A. L. Hodgkin and A. F. Huxley, "A quantitative description of membrane current and its application to conduction and excitation in nerve," *The Journal of physiology*, vol. 117, no. 4, p. 500, 1952.

[43] P. Kanerva, *Sparse distributed memory*. MIT press, 1988.

[44] L. Ge and K. K. Parhi, "Classification using hyperdimensional computing: A review," *IEEE Circuits and Systems Magazine*, vol. 20, no. 2, pp. 30–47, 2020.

[45] A. Rahimi, S. Datta, D. Kleyko, E. P. Frady, B. Olshausen, P. Kanerva, and J. M. Rabaey, "High-dimensional computing as a nanoscalable paradigm," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 64, no. 9, pp. 2508–2521, 2017.

[46] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural Computation*, vol. 1, no. 4, pp. 541–551, 1989.

[47] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[48] M. I. Jordan, "Serial order: a parallel distributed processing approach," University of California, San Diego, Tech. Rep., 5 1986.

[49] J. L. Elman, "Finding structure in time," *Cognitive Science*, vol. 14, no. 2, pp. 179–211, 1990.

[50] M. Schuster and K. K. Paliwal, "Bidirectional recurrent neural networks," *IEEE Transactions on Signal Processing*, vol. 45, no. 11, pp. 2673–2681, 1997.

[51] S. Hochreiter, "Investigations into dynamic neural networks," *Diploma, Technical University ät München*, vol. 91, no. 1, 1991.

[52] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE Transactions on Neural Networks*, vol. 5, no. 2, pp. 157–166, 1994.

[53] W. Maass, "Networks of spiking neurons: The third generation of neural network models," *Neural Networks*, vol. 10, no. 9, pp. 1659 – 1671, 1997.

[54] N. Caporale and Y. Dan, "Spike timing–dependent plasticity: A Hebbian learning rule," *Annual Review of Neuroscience*, vol. 31, pp. 25–46, 2008.

[55] G.-Q. Bi and M.-M. Poo, "Synaptic modifications in cultured hippocampal neurons: Dependence on spike timing, synaptic strength, and postsynaptic cell type," *Journal of Neuroscience*, vol. 18, no. 24, pp. 10 464–10 472, 1998.

[56] L. Lapicque, "Recherches quantitatives sur léxcitation electrique des nerfs traitee comme une polarization," *Journal de Physiologie et de Pathologie Generalej*, vol. 9, pp. 620–635, 1907.

[57] W. Gerstner, "Spike-response model," *Scholarpedia*, vol. 3, no. 12, p. 1343, 2008, revision #91800.

[58] H. Jang, O. Simeone, B. Gardner, and A. Gruning, "An introduction to probabilistic spiking neural networks: Probabilistic models, learning rules, and applications," *IEEE Signal Processing Magazine*, vol. 36, no. 6, pp. 64–77, 2019.

[59] G. Srinivasan, C. Lee, A. Sengupta, P. Panda, S. S. Sarwar, and K. Roy, "Training deep spiking neural networks for energy-efficient neuromorphic computing," in *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2020, pp. 8549–8553.

[60] S. R. Kulkarni and B. Rajendran, "Spiking neural networks for handwritten digit recognition - supervised learning and network optimization," *Neural Networks*, vol. 103, pp. 118 – 127, 2018.

[61] G. Gallego *et al.*, "Event-based vision: A survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 1–1, 2020.

[62] M. Pfeiffer and T. Pfeil, "Deep learning with spiking neurons: Opportunities and challenges," *Frontiers in Neuroscience*, vol. 12, p. 774, 2018.

[63] F. Ponulak and A. Kasiński, "Supervised learning in spiking neural networks with ReSuMe: Sequence learning, classification, and spike shifting," *Neural Computation*, vol. 22, no. 2, pp. 467–510, 2010.

[64] A. Taherkhani, G. Cosma, and T. M. McGinnity, "Optimization of output spike train encoding for a spiking neuron based on its spatio–temporal input pattern," *IEEE Transactions on Cognitive and Developmental Systems*, vol. 12, no. 3, pp. 427–438, 2020.

[65] Y. Liu, W. Zhang, and P. Li, "Enabling non-hebbian learning in recurrent spiking neural processors with hardware-friendly on-chip intrinsic plasticity," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 9, no. 3, pp. 465–474, 2019.

[66] N. K. Kasabov, "NeuCube: A spiking neural network architecture for mapping, learning and understanding of spatio-temporal brain data," *Neural Networks*, vol. 52, pp. 62 – 76, 2014.

[67] Y. Cao, Y. Chen, and D. Khosla, "Spiking deep convolutional neural networks for energy-efficient object recognition," *International Journal of Computer Vision*, vol. 113, no. 1, pp. 54–66, May 2015.

[68] N. Anwani and B. Rajendran, "NormAD - normalized approximate descent based supervised learning rule for spiking neurons," in *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*, 2015, pp. 1–8.

[69] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems (NIPS)*, 2012, pp. 1097–1105.

[70] C. Szegedy *et al.*, "Going deeper with convolutions," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE Computer Society, 2015, pp. 1–9.

[71] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proceedings of the International Conference on Machine Learning (ICML)*, ser. ICML'10. Madison, WI, USA: Omnipress, 2010, p. 807–814.

[72] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," University of Toronto, Tech. Rep., 2009.

[73] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Representation*, vol. 15, no. 1, p. 1929–1958, Jan. 2014.

[74] D. Steinkraus, I. Buck, and P. Y. Simard, "Using GPUs for machine learning algorithms," in *Proceedings of the International Conference on Document Analysis and Recognition (ICDAR)*, 2005, pp. 1115–1120 Vol. 2.

[75] R. Raina, A. Madhavan, and A. Y. Ng, "Large-scale deep unsupervised learning using graphics processors," in *Proceedings of the International Conference on Machine Learning (ICML)*. New York, NY, USA: Association for Computing Machinery, 2009, p. 873–880.

[76] D. C. Cireşan, U. Meier, L. M. Gambardella, and J. Schmidhuber, "Deep, big, simple neural nets for handwritten digit recognition," *Neural Computation*, vol. 22, no. 12, pp. 3207–3220, 2010.

[77] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2015.

[78] R. K. Srivastava, K. Greff, and J. Schmidhuber, "Highway networks," *arXiv preprint arXiv:1505.00387*, 2015.

[79] ——, "Training very deep networks," in *Advances in Neural Information Processing Systems (NIPS)*, 2015, pp. 2377–2385.

[80] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 3431–3440.

[81] O. Ronneberger, P. Fischer, and T. Brox, "U-Net: Convolutional networks for biomedical image segmentation," in *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*. Springer International Publishing, 2015, pp. 234–241.

[82] M. Drozdzal, E. Vorontsov, G. Chartrand, S. Kadoury, and C. Pal, "The importance of skip connections in biomedical image segmentation," in *Deep Learning and Data Labeling for Medical Applications*. Cham: Springer International Publishing, 2016, pp. 179–187.

[83] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, p. 1735–1780, Nov. 1997.

[84] F. A. Gers and J. Schmidhuber, "Recurrent nets that time and count," in *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*, vol. 3, 2000, pp. 189–194 vol.3.

[85] F. A. Gers, J. A. Schmidhuber, and F. A. Cummins, "Learning to forget: Continual prediction with LSTM," *Neural Computing*, vol. 12, no. 10, p. 2451–2471, Oct. 2000.

[86] A. Graves, M. Liwicki, S. Fernández, R. Bertolami, H. Bunke, and J. Schmidhuber, "A novel connectionist system for unconstrained handwriting recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 31, no. 5, pp. 855–868, 2009.

[87] A. Graves, "Supervised sequence labelling," in *Supervised sequence labelling with recurrent neural networks*. Springer, 2012, pp. 5–13.

[88] J. Chung, Ç. Gülçehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *arXiv preprint arXiv:1412.3555*, 2014.

[89] H. Bourlard and Y. Kamp, "Auto-association by multilayer perceptrons and singular value decomposition," *Biological Cybernetics*, vol. 59, pp. 291–294, 1988.

[90] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[91] M. Ranzato, C. Poultney, S. Chopra, and Y. LeCun, "Efficient learning of sparse representations with an energy-based model," in *Advances in Neural Information Processing Systems (NIPS)*. MIT Press, 2007, pp. 1137–1144.

[92] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, 2013.

[93] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol, "Extracting and composing robust features with denoising autoencoders," in *Proceedings of the International Conference on Machine Learning (ICML)*. New York, NY, USA: Association for Computing Machinery, 2008, p. 1096–1103.

[94] Y. Jiang, H. Kim, H. Asnani, S. Kannan, S. Oh, and P. Viswanath, "Turbo autoencoder: Deep learning based channel codes for point-to-point communication channels," in *Advances in Neural Information Processing Systems (NIPS)*, 2019, pp. 2758–2768.

[95] M. Gori, G. Monfardini, and F. Scarselli, "A new model for learning in graph domains," in *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*, vol. 2, 2005, pp. 729–734 vol. 2.

[96] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE Transactions on Neural Networks*, vol. 20, no. 1, pp. 61–80, 2009.

[97] J. Zhou *et al.*, "Graph neural networks: A review of methods and applications," *arXiv preprint arXiv:1812.08434*, 2018.

[98] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun, "Spectral networks and locally connected networks on graphs," *arXiv preprint arXiv:1312.6203*, 2013.

[99] M. Henaff, J. Bruna, and Y. LeCun, "Deep convolutional networks on graph-structured data," *arXiv preprint arXiv:1506.05163*, 2015.

[100] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," in *Advances in Neural Information Processing Systems (NIPS)*, 2016, pp. 3844–3852.

[101] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2017.

[102] D. K. Duvenaud, D. Maclaurin, J. Iparraguirre, R. Bombarell, T. Hirzel, A. Aspuru-Guzik, and R. P. Adams, "Convolutional networks on graphs for learning molecular fingerprints," in *Advances in Neural Information Processing Systems (NIPS)*, 2015, pp. 2224–2232.

[103] J. Atwood and D. Towsley, "Diffusion-convolutional neural networks," in *Advances in Neural Information Processing Systems (NIPS)*, 2016, pp. 1993–2001.

[104] W. L. Hamilton, R. Ying, and J. Leskovec, "Representation learning on graphs: Methods and applications," *IEEE Data Engineering Bulletin*, vol. 40, no. 3, pp. 52–74, 2017.

[105] M. Niepert, M. Ahmed, and K. Kutzkov, "Learning convolutional neural networks for graphs," in *Proceedings of the International Conference on Machine Learning (ICML)*, vol. 48. JMLR.org, 2016, pp. 2014–2023.

[106] F. Monti, D. Boscaini, J. Masci, E. Rodolà, J. Svoboda, and M. M. Bronstein, "Geometric deep learning on graphs and manifolds using mixture model CNNs," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 5425–5434.

[107] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Advances in Neural Information Processing Systems (NIPS)*, 2017, pp. 1024–1034.

[108] F. Gama, E. Isufi, G. Leus, and A. Ribeiro, "Graphs, convolutions, and neural networks," 2020.

[109] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2018.

[110] A. G. Howard *et al.*, "MobileNets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[111] X. Zhang, X. Zhou, M. Lin, and J. Sun, "ShuffleNet: An extremely efficient convolutional neural network for mobile devices," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[112] J. Frankle and M. Carbin, "The lottery ticket hypothesis: Finding sparse, trainable neural networks," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2019.

[113] N. P. Jouppi *et al.*, "In-datacenter performance analysis of a Tensor Processing Unit," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*. New York, NY, USA: Association for Computing Machinery, 2017, p. 1–12.

[114] NVIDIA, "NVIDIA DGX-1 With Tesla V100 System Architecture white paper," Tech. Rep., 2017.

[115] Xilinx, "Accelerating DNNs with Xilinx Alveo accelerator cards," Tech. Rep., 2018.

[116] P. A. Merolla *et al.*, "A million spiking-neuron integrated circuit with a scalable communication network and interface," *Science*, vol. 345, no. 6197, pp. 668–673, 2014.

[117] S. B. Furber, F. Galluppi, S. Temple, and L. A. Plana, "The SpiNNaker project," *Proceedings of the IEEE*, vol. 102, no. 5, pp. 652–665, 2014.

[118] M. Davies *et al.*, "Loihi: A neuromorphic manycore processor with on-chip learning," *IEEE Micro*, vol. 38, no. 1, pp. 82–99, 2018.

[119] C. Deng, S. Liao, Y. Xie, K. K. Parhi, X. Qian, and B. Yuan, "PermDNN: Efficient compressed DNN architecture with permuted diagonal matrices," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, 2018, pp. 189–202.

[120] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally, "EIE: Efficient inference engine on compressed deep neural network," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*. IEEE Press, 2016, p. 243–254.

[121] V. Sze, Y. Chen, T. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2295–2329, 2017.

[122] A. Auten, M. Tomei, and R. Kumar, "Hardware acceleration of graph neural networks," in *Proceedings of the Design Automation Conference (DAC)*, 2020.

[123] T. Geng *et al.*, "AWB-GCN: A graph convolutional network accelerator with runtime workload rebalancing," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, 2020.

[124] N. Unnikrishnan and K. K. Parhi, "A gradient-interleaved scheduler for energy-efficient backpropagation for training neural networks," *arXiv preprint arXiv:2002.05529*, 2020.

[125] N. Gamboa, K. Kudrolli, A. Dhoot, and A. Pedram, "Campfire: Compressible, regularization-free, structured sparse training for hardware accelerators," 2020.

[126] S. Ambrogio *et al.*, "Equivalent-accuracy accelerated neural-network training using analogue memory," *Nature*, vol. 558, no. 7708, pp. 60–67, Jun 2018.

[127] B. L. Deng, G. Li, S. Han, L. Shi, and Y. Xie, "Model compression and hardware acceleration for neural networks: A comprehensive survey," *Proceedings of the IEEE*, vol. 108, no. 4, pp. 485–532, 2020.

[128] K. Xu *et al.*, "Show, attend and tell: Neural image caption generation with visual attention," in *Proceedings of the International Conference on Machine Learning (ICML)*, F. Bach and D. Blei, Eds., vol. 37. Lille, France: PMLR, Jul 2015, pp. 2048–2057.

[129] A. Vaswani *et al.*, "Attention is all you need," in *Advances in Neural Information Processing Systems (NIPS)*, 2017, pp. 5998–6008.

[130] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2017.

Keshab K. Parhi (Fellow, IEEE) received the B.Tech. Degree from the Indian Institute of Technology (IIT), Kharagpur, in 1982, the M.S.E.E. degree from the University of Pennsylvania, Philadelphia, in 1984, and the Ph.D. degree from the University of California, Berkeley, in 1988.

He has been with the University of Minnesota, Minneapolis, since 1988, where he is currently Distinguished McKnight University Professor and Edgar F. Johnson Professor of Electronic Communication in the Department of Electrical and Computer Engineering. He has published over 650 papers, is the inventor of 31 patents, and has authored the textbook VLSI Digital Signal Processing Systems (Wiley, 1999). His current research addresses VLSI architecture design of machine learning systems, hardware security, data-driven neuroscience and molecular/DNA computing.

Dr. Parhi is the recipient of numerous awards including the 2017 Mac Van Valkenburg award and the 2012 Charles A. Desoer Technical Achievement award from the IEEE Circuits and Systems Society, the 2003 IEEE Kiyo Tomiyasu Technical Field Award, and a Golden Jubilee medal from the IEEE Circuits and Systems Society in 2000. He served as the Editor-in-Chief of the IEEE Trans. Circuits and Systems, Part-I during 2004 and 2005. He was elected a Fellow of the American Association for the Advancement of Science (AAAS) in 2017.

Nanda K. Unnikrishnan (Student member, IEEE) is currently pursuing a Ph.D. degree in electrical engineering at the University of Minnesota, Minneapolis, USA. He received his B.Tech in electronics and communication from National Institute of Technology, Karnataka (NITK), Suratkal, India in 2014, and his M.S. in electrical engineering from the University of Minnesota in 2018.

He worked at SilabTech, India (now Synopsys) from 2014 to 2016 as a design verification engineer for mixed-signal designs. He has interned at Qualcomm Technologies Inc, San Diego in Summer 2017 and with Intel Labs in Summer 2018. His research interests lie in design of VLSI architectures for machine learning systems.