Journal of Artificial Intelligence and Cyber Security (JAICS)
An International Open Access, Peer-Reviewed, Refereed Journal

NEUROMORPHIC COMPUTING

[1]Prof P.T.Talole, [2]Mr.Nilesh.M.Jadhav,[3]Mr. Ajay.S.Ingle,[4]Mr.Samyak.G.Sonone, [5]Miss.Divyani.V.Patil

[1] *Asst.Prof, Department of Information Technology Engineering, Anuradha Engineering College, Chikhli*
[2] *Student, Department of Information Technology Engineering, Anuradha Engineering College, Chikhli*
[3] *Student, Department of Information Technology Engineering, Anuradha Engineering College, Chikhli*
[4] *Student, Department of Information Technology Engineering, Anuradha Engineering College, Chikhli*
[5] *Student, Department of Information Technology Engineering, Anuradha Engineering College, Chikhli*

[1]prmod.talole@aecc.ac.in,[2]ingleajay03@gmail.com,[3]nileshjadhav2213@gmail.com ,
[4]samyaksonone9356@gmail.com,[5]patildivyani1230@gmail.com

Abstract: Neuromorphic computing represents a revolutionary approach to artificial intelligence inspired by the architecture and functionality of the human brain. Unlike traditional computing paradigms that rely on binary logic and centralized processing units, neuromorphic systems leverage massively parallel networks of artificial neurons and synapses to mimic the brain's cognitive processes in real time. This seminar explores the foundational principles, current advancements, and future prospects of neuromorphic computing. Topics covered include the biological basis of neural computation, the design and implementation of neuromorphic hardware, applications in machine learning and robotics, as well as challenges such as scalability and energy efficiency. By bridging the gap between neuroscience and computer science, neuromorphic computing holds promise for achieving unprecedented levels of computational efficiency and cognitive capabilities, paving the way towards the next generation of intelligent systems. We survey the current status of Neuromorphic computing applications in real-world and discuss its future.

Keywords: AI, artificial neurons, Neuroscience, biological, Application of Neuromorphic computing in real-world

I. Introduction

Nowadays, neuromorphic computing has become a popular architecture of choice instead of von Neumann computing architecture for applications such as cognitive processing. Based on highly connected synthetic neurons and synapses to build biologically inspired methods, which to achieve theoretical neuroscientific models and challenging machine learning techniques using. The von Neumann architecture is the computing standard predominantly for machines. However, it has significant differences in organizational structure, power requirements, and processing capabilities relative to the working model of the human brain. Therefore, neuromorphic calculations have emerged in recent years as an auxiliary architecture for the von Neumann system. Neuromorphic calculations are applied to create a programming framework. The system can learn and create applications from these computations to simulate neuromorphic functions. These can be defined as neuro- inspired

models, algorithms and learning methods, hardware and equipment, support systems and applications. Neuromorphic architectures have several significant and special requirements, such as higher connection and parallelism, low power consumption, memory collocation and processing. Its strong ability to execute complex computational speeds compared to traditional von Neumann architectures, saving power and smaller size of the footprint. These features are the bottleneck of the von Neumann architecture, so the neuromorphic architecture will be considered as an appropriate choice for implementing machine learning algorithms.

A. What is neuromorphic computing?

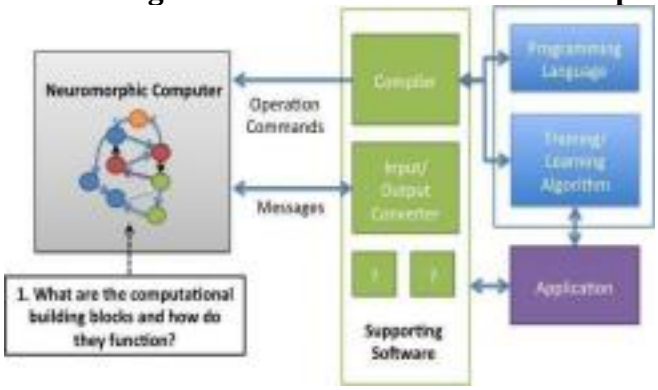
Neuromorphic computing combines computing fields such as machine learning and artificial intelligence with cutting-edge hardware development and materials science, as well as ideas from neuroscience. In its original incarnation, “neuromorphic” was used to refer to custom devices/chips that included analog components and mimicked biological neural activity [Mead1990]. Today, neuromorphic computing has broadened to include a wide variety of software and hardware components, as well as materials science, neuroscience, and computational neuroscience research.

B. Why now?

In 1978, Backus described the von Neumann bottleneck [Backus1978]: In the von Neumann architecture, memory and computation are separated by a bus, and both the data for the program at hand as well as the program itself has to be transferred from memory to a central processing unit (CPU). As CPUs have grown faster, memory access and transfer speeds have not improved at the same scale [Hennessy2011]. Moreover, even CPU performance increases are slowing, as Moore’s law, which states that the number of transistors on a chip doubles roughly every 2 years, is beginning to slow (if not plateau). Though there is some argument as to whether Moore’s law has actually come to an end, there is a consensus that Dennard scaling, which says that as transistors get smaller that power density stays constant, ended around 2004 [Shalf2015]. As a consequence, energy consumption on chips has increased as we continue to add transistors. While we are simultaneously experiencing issues

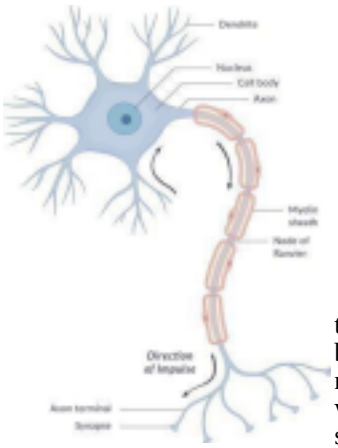
associated with the von Neumann bottleneck, the computation-memory gap, the plateau of Moore's law, and the end of Dennard scaling, we are gathering data in greater quantities than ever before. Data comes in a variety of forms and is gathered in vast quantities through a plethora of mechanisms, including sensors in real environments, by companies, organizations, and governments and from scientific instruments or simulations. Much of this data sits idle in storage, is summarized for a researcher using statistical techniques, or is thrown away completely because current computing resources and associated algorithms cannot handle the scale of data that is being gathered. Moreover, beyond intelligent data analysis needs, as computing has developed, the types of problems we as users want computers to solve have expanded. In particular, we are expecting more and more intelligent behavior from our systems. These issues and others have spurred the development of non–von Neumann architectures. In particular, the goal of pursuing new architectures is not to find a replacement for the traditional von Neumann paradigm but to find architectures and devices that can complement the existing paradigm and help to address some of its weaknesses. Neuromorphic architectures are one of the proposed complement architectures for several reasons: 1. Co-located memory and computation, as well as simple communication between components, can provide a reduction in communication costs. 2. Neuromorphic architectures often result in lower power consumption (which can be a result of analog or mixed analog-digital devices or due to the event-driven nature of the systems). 3. Common data analysis techniques, such as neural networks, have natural implementations on neuromorphic devices and thus are applicable to many "big data" problems. Neuromorphic Computing Architectures, Models, and Applications 4. By building the architecture using brain-inspired components, there is potential that a type of intelligent behavior will emerge. Overall, neuromorphic computing offers the potential for enormous increases in computational efficiency as compared to existing architecture in domains like big data analysis, sensory fusion and processing, real-world/real time controls (e.g., robots), cyber security, etc. Without neuromorphic computing as part of the future landscape of computing, these applications will be very poorly served.

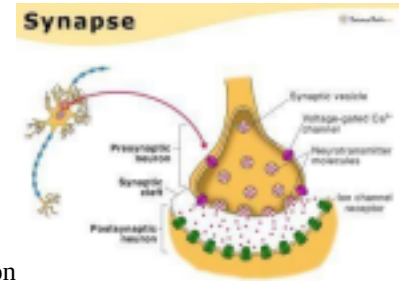
II. Basic computational building blocks and general architectures of neuromorphic systems



The basic computational building blocks of most neuromorphic systems are neurons and synapses. Some neuromorphic systems go further and include notions of axons, dendrites, and other neural structures, but in general, neurons and synapses are the key components of the majority of neuromorphic systems. The information propagation usually is conducted through spikes: whenever enough charge has flowed in at synapses, a neuron generates outgoing spikes, which causes charge to be injected into the post-synaptic neuron.

What other biological components may be necessary for a working neuromorphic system?





Neuron

Beyond neurons and synapses, there are a variety of other biologically inspired mechanisms that may be worth considering as computational primitives. Astrocytes are glial cells in biological brains that act as regulatory systems; in particular, they can stimulate, calm, synchronize, and repair neurons. Certainly we will want to consider what computational effects these regulatory-type systems will have on neuromorphic models. Neurotransmitters and neuromodulator systems also have a significant effect on the behavior of biological brains.

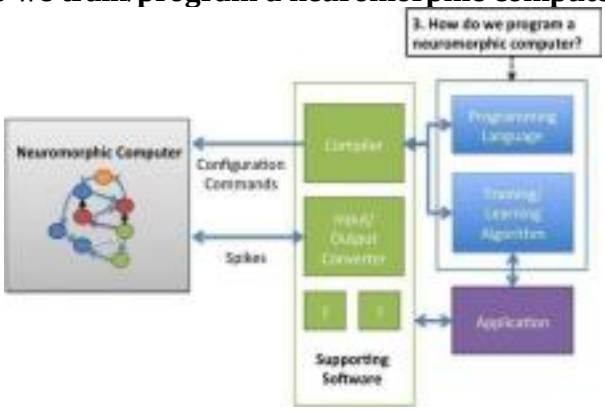
It is not clear how these systems influence the capabilities of biological brains, such as learning, adaptation, and fault

tolerance, but it is worthwhile to consider their inclusion in neuromorphic models. In general, the goalshould be to make minimalistic systems first and then to grow their complexity as we understand how the systems operate together.

How should neurons and synapses be organized for effective information propagation and communication?

Connectivity and plasticity in synaptic weighting dominate the complexity of neuromorphic computing. Offering the compatible connection density as human brain usually results in unaffordable complexity and unsalable system design. Therefore, how to organize neurons and synapses to obtain the greatest density of interconnect at the local level while offering scalable long-range connectivity and balancing the traffic in routing of neural events remains an open research question.

III. How do we train/program a neuromorphic computer



A key question for neuromorphic computers is how to use the device for real applications. In answering this question, we must define what it means to "program" a neuromorphic device, or perhaps more appropriately, how neuromorphic devices will learn or be trained. As such, we define three terms to describe the operation of neuromorphic computers on real applications.

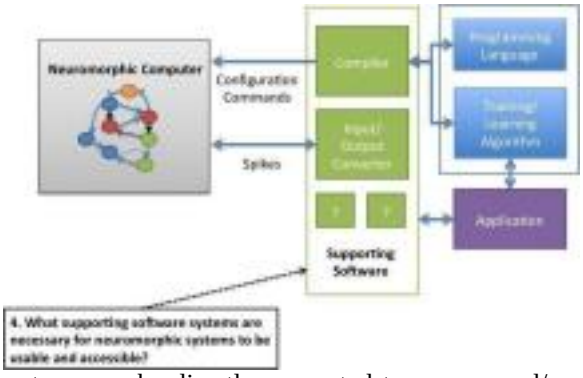
Programmed: The user explicitly setting all parameters and potentially the structure of the networks to perform a particular task.

Trained: A training algorithm is defined. Example situations from the application are presented as part of the algorithm. The user provides feedback (as part of the algorithm) as to how well the neuromorphic device is performing the task, and the algorithm updates parameters and potentially structure based on that feedback. In machine learning this is often referred to as "supervised learning." An example of a training algorithm for certain types of neuromorphic implementations and models is back- propagation. We also categorize reinforcement learning as a training method because the algorithm is receiving feedback. In this case, feedback is either "good" or "bad" as opposed to the correct answer when a wrong answer is given. the algorithm defines ways in which the structure and parameters are updated based on the input it receives. In this case, the user does not provide feedback, but the environment may provide some sort of inherent "reward" or "punishment" (in the case of reinforcement learning). In machine learning, this is often referred to as "unsupervised learning." Unsupervised learning algorithms typically create a compressed "representation" of the input structure. An example of a learning algorithm for certain types of neuromorphic implementations and models is spike- timing- dependent plasticity.

For neuromorphic computing in the real world, there are roles for all three use-cases, and it is likely that some combination of the three will be used. The role of a software developer for neuromorphic computers is going to be radically different depending on the selected programming paradigm. The developer for an explicitly programmed neuromorphic computer will need to explicitly set all parameters and structure and understand the implications of each selection and how they interact. Developers for trained neuromorphic computers will need to consider what examples should be presented and what feedback to provide as part of the training process. They will also likely define parameters for the algorithm itself. For learning neuromorphic computers, the "developer" may be defined as the person who is presenting examples, or there may be no developer at all. For trained and learning neuromorphic computers, the term developer may also be used for the person who *defines* the training or learning algorithm. Another role of a programmer for a neuromorphic computer will be to determine what inputs are given, how those inputs are represented, and how they are presented to the device

IV. What supporting software systems are necessary for neuromorphic systems to be usable and accessible

In order for neuromorphic systems to be a valid complementary architecture for the future computing landscape, it is vital that we consider the accessibility and usability of the systems during the early stages of development. One of the major questions associated with usability and accessibility is how neuromorphic systems will actually be integrated into environments and what supporting software is necessary. We describe two example use-cases of neuromorphic computers or processors and the supporting software that will be required for each use-case



In one use-case, a neuromorphic system may be directly connected to sensors and/or control mechanisms as an embedded system on autonomous vehicles, sensors, or robots that are deployed in real environments. In this case, it will be necessary to build custom protocols and schemes for communication among custom devices within the deployed system itself, as well as the functionality to communicate results and/or data with a centralized server (Figure 10). For this case, it is almost certain that most training/learning will be done off-line (and perhaps also off-chip) and loaded onto the neuromorphic device, so training/learning software will be required. The neuromorphic processor will probably be customized for the particular application with very little programmability, adaptability, and on- line learning, in order to reduce the complexity and, as a consequence, the size and the energy consumption of the device.

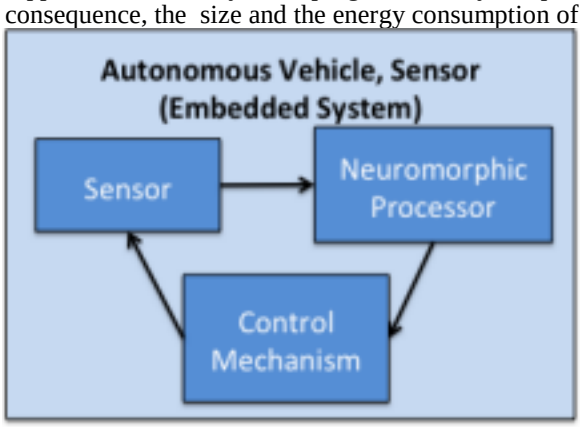


Figure : Embedded system example

A second use-case for neuromorphic computers is as a co-processor in a future heterogeneous node. Figure 11 shows an example of a heterogeneous node, which includes traditional CPUs, GPUs, a neuromorphic processor, a quantum processor, and potentially other emerging architectures. For this use- case, the supporting software will be extensive, and the device itself will likely be more programmable than neuromorphic devices for other use

cases in order to enable the device to be as useful as possible in the heterogeneous node. Communication protocols will be required

V. What does a "program" for a neuromorphic computer look like

Instances of abstract network representations can be thought of as high-level "programs" for neuromorphic devices, which will then need to be "compiled" for each individual device. For some devices, the abstract representation will likely have a direct conversion process, resulting in a simple compiler. However, depending on the implementation and its associated restrictions in parameter representations or connectivity, an extensive

mapping process may be required. Thus, the development of basic compilers for neuromorphic systems to perform the conversion from abstract network representation to machine code for the neuromorphic device will be an important software component

VI. Applications

In daily life, character recognition has high practical application value in the postal, transportation and document management process, such as the recognition of car numbers and license plates in transportation systems. However, the images captured in the natural environment are often blurred due to the limitations of camera hardware, or uncleared by the font is occluded and worn out. At these points, the complete information

of the character cannot be obtain and identification of noisy characters become a key issue. At present, there are several methods for character recognition, which are mainly divided into a neural network, probability statistical , and fuzzy recognition. The traditional character recognition method cannot recognize well under the condition of noise interference. However, the discrete Hopfield neural network has the function of associative memory, which is reasonable for anti-noise processing. By applying this function, characters can be recognized and satisfactory recognition results can be achieved. Besides, the convergence calculation becomes fast for processing

A. Case Study: Character Recognition: The associative memory can be designed based on the discrete Hopfield neural network concept, and the network can recognize the 10 digits that fall in the range from 0-9. Furthermore, despite any disturbance by certain noise due to the specified range of numbers, still has a good recognition effect by network feedback. At this point, the network is composed of a total of 10 stable states that reach to 0–9 numbers. These numbers are represented by 10×10 matrices and are directly described by the binary matrix. In the 10×10 matrix, the number pixel is represented by 1, and the non-number pixel is defined −1 as blank display.

B. Other Applications: Based on the discrete Hopfield neural network, it has the function of associative memory. In recent years, many researchers have attempted to apply Hopfield network to various fields in order to replace conventional techniques to address the issues, such as water quality evaluation and generator fault diagnosis, and have achieved considerable results by applying aforementioned method. For example, in the Internet of Things (IoT) applications, where multiple links fail and break the real-time transmission services, and due to this reason, the fault cannot be quickly located at that particular point. The relationship between the fault set and the alarm set can be established through the network topology information and the transmission service, which is compatible with the proposed Hopfield Neural Network. The built-in Hopfield algorithm of the energy function is used to resolve fault location, and hence, it is found that integration of aforementioned algorithm with the

IoT will improve transmission services in smart cities.

VII. Future Plan and Challenges

For the future of neuromorphic chips, it is the key to break through the development direction of von Neumann's structure limitations. Because the basic operations of neural networks are the processing of neurons and synapses, the conventional processor instruction set (including ×86 and ARM, etc.) was developed for general- purpose computing. These operations are arithmetic operations (addition, subtraction, multiplication and division) and logical operations (AND-OR-NOT) . It often requires hundreds or thousands of instructions to complete the processing of neuron computing, making the low processing efficiency of the hardware inefficient.

Currently, neural computing needs a completely different design than the vonNeumann architecture. The storage and processing are integrated into the neural network, whereas in von Neumann's structure, there it is separated and realized respectively by memory and computational unit. There is a huge difference between the two computing when using current classical computers based on the von Neumann architecture (such as CPUs and GPUs) to run neural network applications. They are inevitably restricted by a separate storage and handling

structure, which has caused a lower efficiency over the impacts. Although the current FPGA and ASIC can meet the requirements of some neural network application scenarios, a new generation of architecture like neuromorphic chips and integrated computing design will be used as the underlying architecture to improve the neural network computing in the long-term planning.

There are still many problems in the research of new materials for the neuromorphic hardware. In the future, researchers in the neuromorphic disciplines consider new materials belonging to neuromorphic computing can be found

in place of transistors to new hardware design. For example, the array composed of memristor that is a plastic element can be stored and processed to integrate for the neuromorphic hardware. It has a high switching current ratio, a light effective mass, a large adjustable band gap and large electron mobility, which provides a favorable basis for successful preparation of low-power neuromorphic hardware.

Eventually, the architecture, algorithm and programming scheme of adaptive neuromorphic computing is in a wide blank and a long way to reach a final goal that replaces to von Neumann's structure in the artificial intelligence discipline. But the frontiers of neuromorphic computing knowledge are being pushed farther outwards over the time, and the future opens a bright prospect.

VIII. Conclusion :

Although neuromorphic computing has gained widespread attention in recent years, however, it is still considered to be in the infancy stage. The existing solutions mostly focus on a single application at the hardware or software level, and majority of them are only suitable for handling limited applications. In addition, there are many software-based neural network applications that has been deployed, but hardware-based neural network design has been the key to the neuromorphic design. Convention neural network circuit implementation is thought of time-consuming and inconvenient. In order to apply a simple and fast design method to neural network hardware, which can optimize and manufacture neuromorphic computing systems, needs to

systematically unificate the requirements of the software calculation process. Furthermore, it can process and improves the final software-level application indicators to quantify hardware attributes. Finally, a testable solution for a specification component can be achieved.

IX. REFERENCES

1. [Backus1978] Backus, John. "Can programming be liberated from the von Neumann style?: a functional style and its algebra of programs." Communications of the ACM21.8 : 613–641.

2. [Chen2012] Tianshi Chen, Yunji Chen, Marc Duranton, Qi Guo, Atif Hashmi, Mikko Lipasti, Andrew Nere, Shi Qiu, Michele Sebag, Olivier Temam. "BenchNN: On the Broad Potential Application Scope of Hardware Neural Network Accelerators," Proceedings of the IEEE International

3. [Hennessy2011] Hennessy, John L., and David A. Patterson. Computer architecture: a quantitative approach. Elsevier.

4. [WSP:Taha] Tarek Taha, Raqibul Hasan, and Chris Yakopcic. "Energy Efficiency and Throughput of Multicore Memristor Crossbar Based Neuromorphic Architectures."

5. [WSP:Williams] R. Stanley Williams. "A Nanotechnology-Inspired Grand Challenge for Future Computing."

6. [WSP:Wolfe] Chris Carothers, Noah Wolfe, Prasanna Date, Mark Plagge, and Jim Hendler. "Large-Scale Hybrid Neuromorphic HPC Simulations, Algorithms and Applications."