Neuromorphic Computing with Spiking Neural Networks

David L. Escudero
lucht013@morris.umn.edu
Division of Science and Mathematics
University of Minnesota, Morris
Morris, Minnesota, USA

Abstract

Neural networks are a popular tool for machine learning. Because of the underlying architecture of modern computer design, neural networks are beginning to face mounting issues of power consumption and performance inefficiency due to implementation. This paper will provide background on neuromorphic computing, an area of research aiming to develop more efficient neural networks utilizing biological inspiration. It explores neuromorphic computing through background and discussion of a specific type of neural network known as a spiking neural network, including difficulties with training and a solution proposed by Ledinauskas et. al. In addition, I introduce a hardware implementation of spiking neural networks using components called memristors.

Keywords: neuromorphic computing, von Neumann bottleneck, artificial neural network, spiking neural network, memristor

1 Introduction

There is a growing demand for machine-based solutions to abstract problems such as computer vision, natural language processing, and driving automation. Neural networks have come to be one of the promising approaches to solving these problems. Unfortunately, neural networks as they currently stand are not ideal for all problems and current computer architectures are not optimized for doing work with them.

Though traditional neural networks are able to solve many abstract problems, they still have nowhere near the adaptability or ease of learning as human beings. Neuromorphic computing, among other things, is an attempt to bridge the gap of technology by taking cues from what we see in biology. Neural networks are only loosely adapted from biological brains; this is in part because of the difficulty in implementing all of the mechanisms at play within biological systems. Spiking neural networks are a more directly biologically-inspired spin on current neural networks. This paper will describe some of the problems of designing these networks as well as a novel approach to developing a more efficient, similar performing network.

One of the solutions to the problem of hardware limiting the effectiveness of neural networks is the discovery and development of memristors, circuit components which change their electrical properties based on prior input. In this paper, I discuss the properties that describe memristors as well as an implementation of memristors being used for spiking neural networks.

2 Background

This section covers the important concepts behind neuromorphic computing.

2.1 Computer Architectures

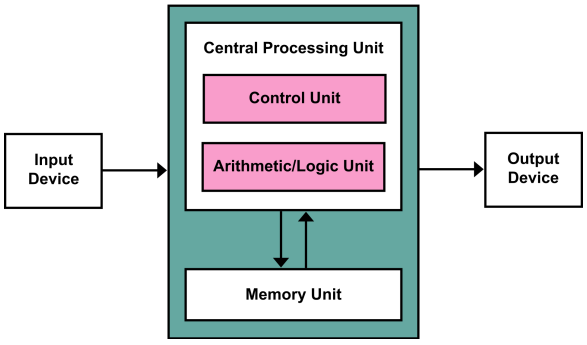

Figure 1. A basic Von Neumann machine [16]

A computer architecture describes the design and implementation of a computer system. This includes how memory is managed and accessed, where instructions occur, and how arithmetic logic is processed. [13] All computers have some form of architecture which guided its design; most computers of today use some variant or expansion of one known as the von Neumann architecture.

Outlined by John von Neumann in 1945, this computer architecture has three main components: shared memory for programs and data, input/output devices, and a central processing unit (CPU) which is divided into a control unit and a logic unit [16]. The control unit manages the interpretation of instructions being received, and the logic unit performs arithmetic and bitwise operations. An example of a basic von Neumann setup can be seen in Figure 1 where input and output devices interface with a central "box" housing a central processing unit and a memory unit.

2.2 Von Neumann Bottleneck

The von Neumann architecture is designed so that program and data memory are part of a shared space separate from the CPU. The CPU receives instructions from memory, performs tasks, then writes back to memory. A problem arises when a lot of data needs processing. Because the calling and writing of memory takes time, the CPU in a modern computer must *idle* while memory is accessed; this limit in data transfer rate is known as the von Neumann bottleneck. This is most apparent when there is a large amount of data which requires a small amount of computation [16].

2.3 Neuromorphic Computing

Neuromorphic computing is an overarching term which describes biologically inspired technology which often does not utilize the von Neumann architecture [11]. Some of the main reasons for doing this are to develop machines which are capable of learning and adapting similarly to humans– as well as finding ways to do this while using less power. This is particularly useful when attempting to create machines which are capable of autonomous driving and processing language.

Examples of neuromorphic computing are memristors and spiking neural networks. Neural networks are a method of machine learning inspired by biological neurons. A related method which more closely resembles human neurons is spiking neural networks, a topic which will be discussed in section 2.7 after covering the basics of conventional neural networks and an important variant used for categorizing and interpreting images known as convolutional neural networks.

2.4 Neural Networks

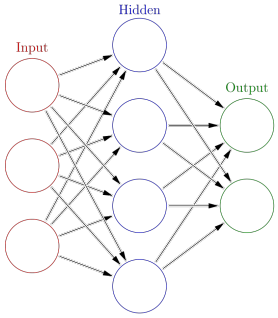

Figure 2. A simple artificial neural network [12]

Neural Networks (NN) are a broad class of algorithms which are loosely modeled after the operating principles of neurons. Networks are built from layers of artificial neurons which connect with one another and are affected by a given input as well as each other, as seen in Figure 2. Neural networks can have multiple hidden layers. The number of layers a network has describes a property known as its *depth*. A deep neural network refers to a NN which has many hidden layers.

Figure 2 represents a simple neural network. Each circle in the diagram represents one neuron. Neurons are divided into three categories, an input layer, a hidden layer, and an output layer. The input layer takes in data, which may come in any form such as images, audio, or text. The input layer then influences the behavior of the hidden layers. The hidden layers finally influence the output layer to deliver a classification. Layers are generally arranged in a manner known as feedforward, where input comes in from the left side and propagates to the right where an output might activate. Often, all neurons in one layer are connected to every other neuron in the following layer; each connection being assigned a weight. These weights correspond to a function which regulates whether or not a neuron will contribute toward the activation of another neuron, called the *activation function*. Often in NN implementations the activation function is continuous, meaning that exact values can be found by taking smaller and smaller approximations. If a neuron is receiving a strong enough cumulative signal from all neurons it is connected to, it will activate and send a signal forward to the following layer. A method by which weights are adjusted to fit a desired model is known as backpropagation and the overall process is known as training. To train a NN, data which has already been classified is fed as input, output classification from the training data is then used in conjunction with the known correct classifications to direct how weights are adjusted. Real NNs may have many neurons in each category and may have multiple hidden layers [9].

2.5 Comparing Neural Networks

Figure 3. Figure showing digits represented within the MNIST dataset [15]

Neural networks are not all built for the same purposes. Part of training NNs is developing them to perform classification faster and more effectively. Before this can be done, there needs to exist a set of standards by which one trained NN can be compared to another. The generally accepted method by which NNs are compared is through the use of

"benchmark" datasets. These datasets are large and deliberately well formatted for a specific purpose. Scoring of these datasets usually is in the form of error rates generated from a test set.

One of the most popular datasets for pattern recognition is the MNIST database [5]. MNIST is a collection of handwritten digits, preformatted for the purposes of machine learning. Figure 3 shows a sampling of examples for human handwriting. Each digit is formatted to be black and white, and stay within the center of a 28x28 pixel box [5]. The aim of this dataset is to train NNs to classify handwritten digits.

Another popular dataset, used for image classification, is the CIFAR-10 dataset. This dataset is made up of 60,000 images of size 32x32 pixels. This images are in color and each image belongs to one of ten possible classes. The classes for the dataset are airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck. Each category contains 6,000 images.

2.6 Convolutional Neural Networks

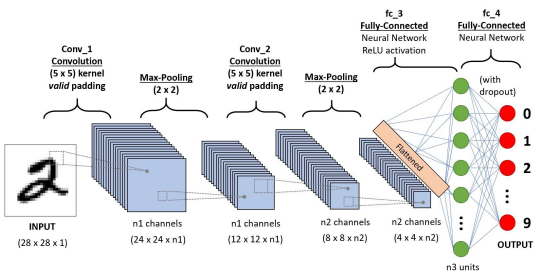

Figure 4. A visualization of a convolutional neural network being used to classify images from the MNIST dataset [10]

Convolutional neural networks (CNN) are a variation of deep neural network which have been found to work particularly well with images and image classification. This is an involved process which does multiple passes across an image using a kernel to retrieve important features from a given image. Features are image properties which are generated by a CNN. Figure 4 is a visualization of how a CNN interprets an image, in this case a number from the MNIST dataset. Starting from the input, there are alternating convolution and pooling layers which get progressively smaller before finishing at a fully connected NN for classification.

Convolution is a name for a process by which functions multiply to create a new, third function. This forms the underpinning of convolutional neural networks. For an image to be processed, an image is broken down into a numerical representation of itself. In the case of Figure 4, the digit shown would be represented by having each pixel assigned a value based on whether or not it has color.

A square mask or "kernel" describes a feature of an image; this could be something like vertical or horizontal lines. When a kernel is "on" a portion of an image, it compares how similar that region is to the feature the kernel describes. It then travels across an image in discrete steps, generally from left to right, up to down, generating a new "image" known as a feature map. Often because a single kernel cannot capture all of the relevant information held within an image, multiple kernels sometimes referred to as "channels" will work through an image to capture multiple details. Feature maps are represented by the blue squares in Figure 4.

After convolution, a process known as *pooling* begins. This process is similar to how a kernel generates a feature map, a smaller kernel travels across a feature map, but here the CNN designer decides how the pooling kernel will probe the feature map. Figure 4 shows examples of *max pooling*. Max pooling would refer to the pooling kernel selecting the highest value in a given step. For example, say we have a 2 pixel by 2 pixel square pooling kernel. Max pooling would dictate that for each step, the highest value within the mask would be the number used for the given coordinate in the resulting feature map.

Once convolution and pooling is complete, the final feature maps are individually connected to input nodes which have their own layers that lead to output. This process of convolution and pooling significantly reduces the required training images needed. Often CNNs are comprised of multiple convolution and pooling layers before final processing of output classification. The primary purpose for this is to only feed the most relevent information to the NN so that processing requirements are reduced. To illustrate this point, take an image who's size is 1920x1080, a standard camera dimension. There are a total of 2,073,600 pixels contained within that image. For image classification without convolution, each pixel would need to be "plugged" into unique input nodes of the NN which would also each need to be fully connected to any subsequent layers, meaning there would be a *minimum* of approximately 4.3 trillion connections for a 1920x1080 pixel image. A network like this would be totally impractical for all but the most powerful computing clusters of today.

2.7 Spiking Neural Networks

Spiking neural networks (SNN) are a newer class of NNs which more closely mimic biological neurons. Neurons operate on a similar principle to traditional NNs, though they have some key differences. In a traditional NN, the activation function being used utilizes a continuous gradient. In an SNN, the activation function is a stream of binary pulses called a *spike train*. This chain of pulses accumulates within a given receiving neuron it and builds a *charge*; this is analogous to how biological neurons accumulate and release electrical potential through neurotransmitters and ions. Once a charge has reached a level called the *membrane threshold*, the neuron fires a spike pulse and the previously accumulated charge falls back to its baseline. These pulses are not persistent and

if a neuron does not receive pulses regularly, it will begin to *decay* and charge will be lost until a baseline level has been reached [8].

Depending on the specific ruleset under which an SNN operates, when neurons fire, the strength of the connection between the sending and receiving neuron strengthens. One of the problems involved in developing SNNs that will be discussed in later sections is training. Because the activation function of SNNs involves spike trains instead of continuous values, spike trains are called nondifferentiable, since pulses are discrete "actions". To account for this, when developing an SNN, inputs must be encoded using a different method such as pulse rate or amplitude as opposed to a unique value.

2.8 Memristors

2.8.1 Circuit Elements. In electrical theory, there are generally four main variables used to describe a circuit, voltage (denoted by V), current (denoted by I), charge (denoted by q), and magnetic flux (denoted by Φ). These all have relationships to one another made apparent by different series of equations. An important equation to understand memristors is known as Ohm's Law: $V = I * R$. We can see that voltage V is equal to the current I multiplied by the resistance R. We can then divide both sides by I to determine the resistance, $R = \frac{v}{I}$. This method of determining resistance is key to memristors as they have varying resistance depending on voltage applied.

2.8.2 Operation. The memristor was first conceived by Leon Chua in 1971. At its most fundamental level, memristors are passive circuit elements which change and "remember" resistance based on the amount of applied voltage. Components which have variable resistance depending on external conditions are not unique to memristors.

An extreme example of a component which shares this property is a fuse. A fuse operates by allowing current at a certain resistance up until a certain voltage, at which point the fuse pops and its resistance becomes infinite. It stays at this new state, "remembering" the current that was passed through before. Another example of this is a variable resistor that is attached to a microcontroller. The microcontroller adjusts the resistance based on the current applied. What makes a memristor unique from components like a fuse or a variable resistor attached to a microcontroller is that the change in resistance is not confined to one direction as in the case of a fuse and requires no active change as in the case of the variable resistor.

2.8.3 Pinched Hysteresis. *Hysteresis* describes how the state of some systems is dependent on its history. Traditional resistors do not have the property of hysteresis; every voltage value has exactly one corresponding current. The line described by the bottom of Figure 6 shows that regardless of how the current changes, there will only be one state which corresponds to it. The top of Figure 6 demonstrates

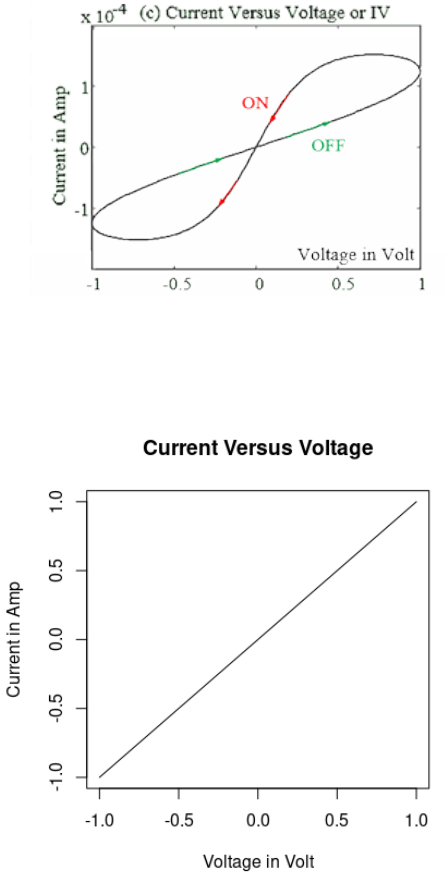

Figure 5. An idealized memristor IV plot (top) alongside a resistor IV plot (bottom) [7](edited)

resistance, and thereby hysteresis, as seen in memristors. The colored "on" and "off" arrows indicate changing voltage in a sinusoidal pattern. As voltage shifts from one value to another, the resulting resistance does not stay the same; it changes based on where it was before [7]. Another important aspect of memristors is that each reachable state along the curve is stable, meaning that if current is not applied, the memristor will not reset its position; any changes that occur will be relative to its state when it was turned off.

2.8.4 Relationship to Neuromorphic Computing. Memristors are a promising avenue for the development of more efficient neuromorphic systems. By linking resistance to activation threshold and current to weight, a hardware-based artificial neuron can be created. This is particularly well suited to the "neurons" used in SNNs. A review paper by Camuñas-Mesa et al. describes how memristor behavior is uniquely similar to the underlying mechanics of SNNs [2].

This similarity facilitates the elimination of processing overhead associated with software implementations in von Neumann machines and overcoming the previously mentioned von Neumann bottleneck [2].

3 Implementations

One of the prominent issues involved with spiking neural networks is how an SNN gets trained; this occurs because spike trains are binary pulses. Normally for training NNs, input is converted into continuous values, for example in the MNIST database, digits are input in the form of greyscale pixels of varying smooth levels. Outside of very simple models, this poses a challenge for SNNs in the form of converting given input data into spike trains. One method for solving this has been to develop a deep neural network which solves the given problem and then manually converting it into an SNN. This is not ideal because doing so does not take advantage of the full potential of SNNs [3].

3.1 Problems with Training

As was stated previously, a critical component for the creation of useful neural networks is training. A standard method for training NNs is back propagation. To train NNs, data from a desired set is used as input for a NN, which will generate a given output; in the case of image classification for MNIST, hand drawn digits would serve as input and a number 0-9 would be the output.'

Results from this process are then compared to the actual results and evaluated basted on how "incorrect" the model performed. Since these networks are really functions, we can craft other functions which model this incorrectness. Through the use of calculus and the knowledge of the performance of the model weights, we can minimize what in this case is known as a *loss function*. This process of back propagation is repeated many times to get the loss function to a "local minima", a process known as gradient descent. A visualization of gradient descent for a simple NN can be seen in Figure 7.

These functions have a property known as *continuity*, which allows gradient descent to occur. A function which is continuous has minimum values which can be approximated by exploring smaller and smaller portions of the local area. Functions which don't have continuity have areas that "jump", which cannot be found via approximation. Spiking neural networks are not naturally continuous because the gradient of different possible values for input has been replaced with discrete spikes. Having spikes for activation makes gradient descent a difficult method to choose.

3.2 Conversion from CNN to SNN

In 2015, Diehl et al. developed a method to convert deep convolutional neural networks into spiking neural nets of similar depth. Prior to this, there was difficulty in developing SNNs due to the large computational cost. It was found that by developing a process of using a uniform activation function for all neurons in a CNN, training the CNN, directly mapping the weight values to an identical SNN, and then using a novel method of normalizing weights, SNN accuracy was nearly on equivalent to the performance of a CNN of similar complexity while using less computational power than prior methods. [3].

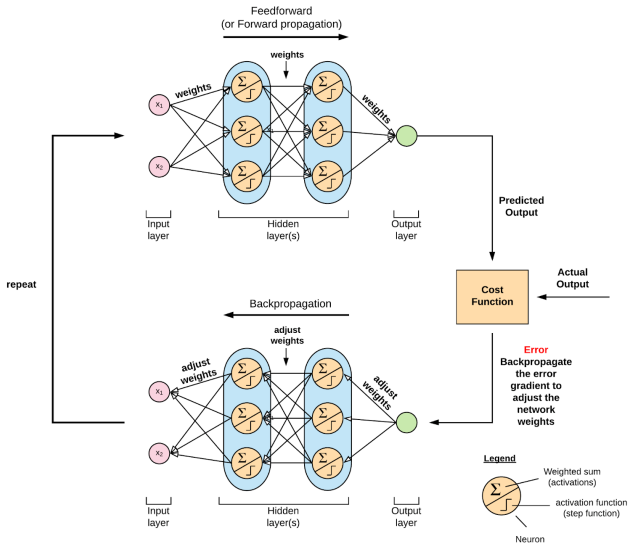

Figure 6. Gradient descent in a traditional NN [1]

3.3 Direct Training of SNN

In 2020, Ledinauskas et al. released a preprint paper which describes a new method for directly training SNNs without the need to start from a trained CNN [6]. Their method relies upon how gradient descent does not require a perfectly accurate gradient. Because the problems of training an SNN stem from spike trains not mapping as functions for use in gradient descent, Ledinauskas et al. use a novel approach which involves something known as a "surrogate gradient" to generate very deep SNNs [6]. This surrogate gradient is an alternative gradient which mimics the nondifferentiable spike activation function. Once this adjustment is made, the SNN can be trained similarly to traditional NNs with back propagation.

Ledinauskas et al. tested their methodology on the MNIST dataset, the CIFAR-10 dataset, and the CIFAR-100 dataset, a variation of CIFAR-10 which has 100 classes of 600 images each [4]. They determined that for the MNIST and CIFAR-10 datasets, that they were able to achieve similar results using direct training of SNNs to other already existing means of training SNNs [6]. Their results can be seen in Table 1.

Model	Method	Acc
Mozafari et al.	reward-modulated STDP	97.2
Tavanaei et al.	STDP + gradient descent	98.6
Lee et al.	back prop (older approach)	99.59
Ledinauskas et al.	**back prop**	**99.40**

Table 1. Table showing accuracy of different SNN approaches on MNIST dataset [6]

Model	Method	Acc
Han et al.	VGG16, ANN-SNN conv.	70.93
Rathi et al.	VGG11, ANN-SNN conv.	70.94
Ledinauskas et al.	**back prop**	**58.5**

Table 2. Table showing accuracy of SSN and different ANN-SNN conversion approaches on CIFAR-100 dataset [6]

Performance on the CIFAR-100 dataset, however, showed that direct training of SNNs had a nontrivial drop in performance relative to NN-SNN conversion, as shown by Table 2. A hypothesis posed by Ledinauskas et al. for the drop in performance is that because the gradient being used is an approximation, inaccuracies begin to accumulate as the network gets sufficiently deep.

Overall, they conclude that although SNNs still underperform in comparison to ANNs and other modern techniques for heavy training requiring deeper layering, it appears that shallower SNNs outperformed similar NNs, possibly demonstrating specialized use in Internet of Things devices. [6].

4 Conclusions

Neuromorphic computing is an emerging area of focus which aims to solve problems in computing with inspiration from biological systems. Emerging implementaions of neuromorphic computation may help reduce power consumption in use of neural networks. SNNs are an emerging neuromorphic NN design which operates differently from traditional NNs. SNNs utilize spiking neurons which accumulate synaptic potential over time to communicate. Though they show promise for using markedly less computational power to train, problems arise when attempting to train them. A novel method of training SNNs directly developed by Ledinauskas et al. demonstrates potential for more efficient NNs. This, in conjunction with bringing SNN principles into hardware using memristors could bring significant power reduction for neural networks.

Acknowledgments

This paper would not be possible without the help and feedback of Peter Dolan and Elena Machkasova, as well as encouragement from family and friends. Thank you.

References

[1] Ekaba Bisong. 2019. A Workshop on Machine Learning using Google Cloud Platform. https://ekababisong.org/ieee-ompi-workshop/

[2] Luis A. Camuñas-Mesa, Bernabé Linares-Barranco, and Teresa Serrano-Gotarredona. 2019. Neuromorphic Spiking Neural Networks and Their Memristor-CMOS Hardware Implementations. *Materials* 12, 17 (2019). https://doi.org/10.3390/ma12172745

[3] P. U. Diehl, D. Neil, J. Binas, M. Cook, S. Liu, and M. Pfeiffer. 2015. Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing. In *2015 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 1–8. https://doi.org/10.1109/IJCNN.2015.7280696

[4] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. 2009. The CIFAR-10 Dataset. https://www.cs.toronto.edu/~kriz/cifar.html

[5] Yann LeCun, Corinna Cortes, and Christopher J.C. Burges. 1998. THE MNIST DATABASE. http://yann.lecun.com/exdb/mnist/

[6] Eimantas Ledinauskas, Julius Ruseckas, Alfonsas Juršėnas, and Giedrius Buračas. 2020. Training Deep Spiking Neural Networks. arXiv:2006.04436 [cs.NE]

[7] S. P. Mohanty. 2013. Memristor: From Basics to Deployment. *IEEE Potentials* 32, 3 (2013), 34–39. https://doi.org/10.1109/MPOT.2012.2216298

[8] Michael Pfeiffer and Thomas Pfeil. 2018. Deep Learning With Spiking Neurons: Opportunities and Challenges. *Frontiers in Neuroscience* 12 (2018), 774. https://www.frontiersin.org/article/10.3389/fnins.2018.00774

[9] Srivignesh Rajan. 2020. An Introduction to artificial neural networks. https://towardsdatascience.com/an-introduction-to-artificial-neural-networks-5d2e108ff2c3

[10] Sumit Saha. 2018. A Comprehensive Guide to Convolutional Neural Networks - the ELI5 way. https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

[11] Catherine Schuman, Thomas Potok, Robert Patton, J. Birdwell, Mark Dean, Garrett Rose, and James Plank. 2017. A Survey of Neuromorphic Computing and Neural Networks in Hardware. (05 2017). https://arxiv.org/abs/1705.06963

[12] Wikipedia. 2021. Artificial neural network — Wikipedia, The Free Encyclopedia. http://en.wikipedia.org/w/index.php?title=Artificial%20neural%20network

[13] Wikipedia. 2021. Computer architecture — Wikipedia, The Free Encyclopedia. http://en.wikipedia.org/w/index.php?title=Computer%20architecture

[14] Wikipedia. 2021. Electrical element — Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/wiki/Electrical_element [Online; accessed 25-March-2021].

[15] Wikipedia. 2021. MNIST database — Wikipedia, The Free Encyclopedia. http://en.wikipedia.org/w/index.php?title=MNIST%20database

[16] Wikipedia. 2021. Von Neumann architecture — Wikipedia, The Free Encyclopedia. http://en.wikipedia.org/w/index.php?title=Von%20Neumann%20architecture