Journal of Artificial Intelligence and Emerging Technologies
Volume 3, Issue 4, pp 42-48, April-2026
ISSN (online): 3049-1967
DOI: https://doi.org/10.47001/JAIET/2026.304006
Available Online at https://www.jaiet.com

AI-Driven Arithmetic Logic Units

[1]Faiqua Rizwan, [2]Priyanjala Yadav, [3]Shikha Yadav, [4]*Mohd. Imran Aziz

[1,2,3,4]Physics Department, Shibli National College, Azamgarh-India
E-mail: azizimran33@gmail.com

Abstract: Traditional neural networks struggle to generalize arithmetic operations beyond the numerical range seen during training. This paper investigates the implementation and evaluation of a Neural Arithmetic Logic Unit (NALU).A specialized neural network module introduced in the paper AI-Driven Arithmetic Logic Units, designed to improve numerical reasoning and arithmetic generalization. The NALU architecture extends the Neural Accumulator (NAC) by incorporating both additive and multiplicative computation pathways, controlled by a learnable gating mechanism. This structure enables the model to learn exact arithmetic operations such as addition, subtraction, multiplication, and division, while maintaining the ability to extrapolate beyond the training range. In this manuscript, the NALU model was implemented using TensorFlow and evaluated on synthetic arithmetic tasks.

Keywords: Neural Networks, Artificial Neural Network, Tensor Flow, Logic Gates.

I. INTRODUCTION

The human brain is an incredibly intricate, nonlinear, and parallel processing system (information processing unit) [1]. It possesses the ability to arrange its structural elements, known as neurons, in order to execute specific calculations (such as pattern recognition, perception, motor control, etc.). These calculations occur at speeds that far surpass the quickest digital computers available today. More specifically, the brain consistently performs perceptual recognition tasks (for instance, identifying a familiar face in an unfamiliar environment) in about 100-200 milliseconds, while tasks of much lesser complexity can take days on a standard computer. When It became clear that the human braiN functions in a completely distinct manner compared to traditional digital computers, research into artificial neural networks (often called "neural networks") was spurred. Generally speaking, a neural network is a system designed to replicate how the brain executes specific tasks [2-5]. This network is typically constructed using electronic components or is simulated via software on a digital computer. Neural networks rely on extensive interconnectivity of "neurons" or processing units that are essential for the functioning of a neural network [6]. In fact, a neural network acts as a highly parallel distributed processor composed of simple processing units, which naturally tends to retain experiential knowledge and make it accessible for application. The network gathers knowledge from its surroundings through a learning process [7-12]. The strengths of connections between neurons, referred to as synaptic weights, are utilized to store the acquired knowledge. Machine Learning algorithms automatically build a mathematical model using sample data also known as 'training data' to make decisions without being specifically programmed to make those decisions. Machine learning is applicable to fields such as Speech recognition, text prediction, handwritten generation, genetic algorithms, artIficial neural networks, natural language processing [13]. To these single-value neurons, we apply operators that are capable of representing simple functions (e.g., +, −, ×, etc.). These operators are controlled by parameters which determine the inputs and operations used to create each output. However, despite this combinatorial character, they are differentiable, making it possible to learn them with back propagation [14-18].

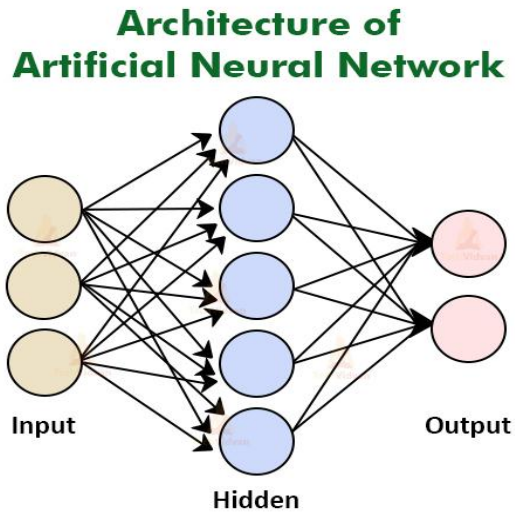

Figure 1: Architecture of ANN

Journal of Artificial Intelligence and Emerging Technologies
Volume 3, Issue 4, pp 42-48, April-2026
ISSN (online): 3049-1967
DOI: https://doi.org/10.47001/JAIET/2026.304006
Available Online at https://www.jaiet.com

II. METHODS OF NEURAL ARITHMETIC

A Neural network architecture comprises a number of neurons or activation units as we call them, and this circuit of units serves their function of finding underlying relationships in data. And it's mathematically proven that neural networks can find any kind of relation/function regardless of its complexity, provided it is deep/optimized enough, that is how much potential it has [19]. Now let's learn to implement a neural network using TensorFlow. TensorFlow is a popular open-source machine learning framework that can be used to implement neural arithmetic's [20]. TensorFlow can be used to define and train the neural network architecture of the arithmetic. This includes defining the layers, activation functions, and optimization algorithms. A neural arithmetic is a neural network that takes multiple input signals and produces a single output signal [21]. The neural arithmetic is trained to learn the complex patterns in the input signals and adaptively select the desired output [22]. The architecture of a neural multiplexer is shown in Figure 2. The input signals are fed into a shared encoder network, which extracts features from the inputs. The features are then fed into a selector network, which selects the desired output [23].

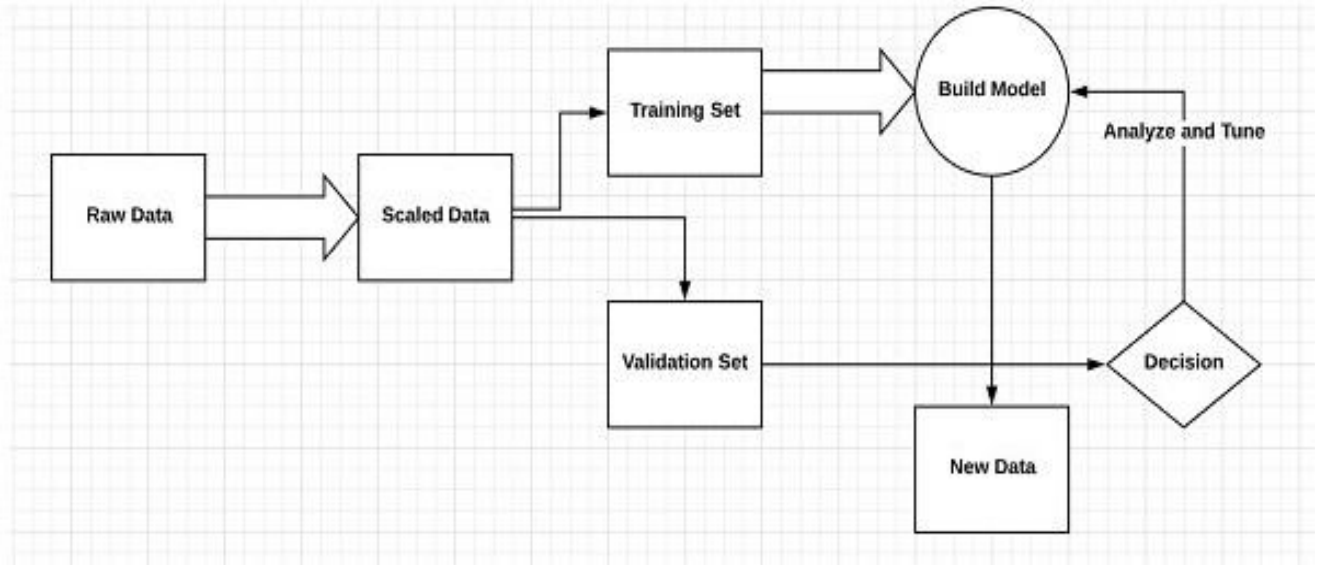

Figure 2: Neural Network Architecture

III. ARTIFICIAL NEURAL NETWORK

Artificial neural networks are systems motivated by the distributed, massively parallel computation in the brain that enables it to be so successful at complex control and classification tasks [24]. The biological neural network that accomplishes this can be mathematically modeled by a weighted, directed graph of highly interconnected nodes (neurons). An ANN initially goes through a training phase where it learns to recognize patterns in data, whether visually, aurally, or textually. During this training phase, the network compares its actual output produced with what it was meant to produce the desired output. The difference between both outcomes is adjusted using a set of learning rules called back propagation. This means the network works backward, going from the output unit to the input units to adjust.

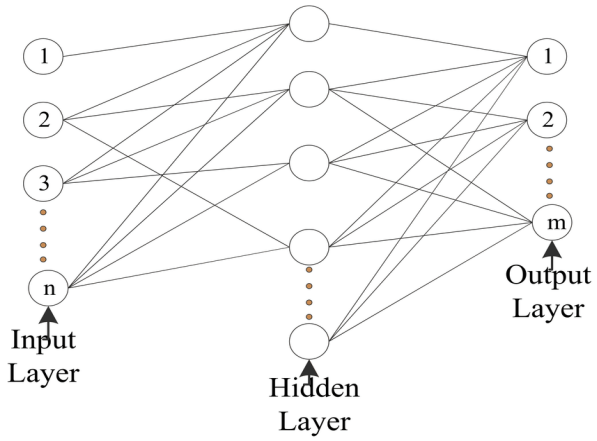

Figure 3: Architecture of typical ANN

Journal of Artificial Intelligence and Emerging Technologies
Volume 3, Issue 4, pp 42-48, April-2026
ISSN (online): 3049-1967
DOI: https://doi.org/10.47001/JAIET/2026.304006
Available Online at https://www.jaiet.com

The weight of its connections between the units until the difference between the actual and desired outcome produces the lowest possible error. The structure of a typical Artificial Neural Network is given in Figure 3. Neuron An ANN has hundreds or thousands of processing units called neurons, which are analogous to biological neurons in human brain. Neurons interconnected by nodes. These processing units are made up of input and output units [25]. The input units receive various forms and structures of information based on an internal weighting system and the neural network attempts to learn about the information presented to it. The neuron multiplies each input $(x_1, x_2, \ldots\ldots, x_n)$.

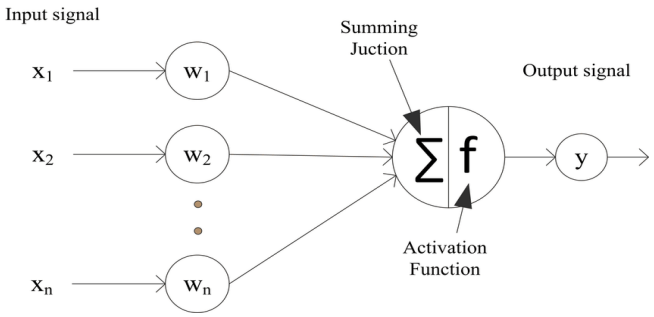

Figure 4: Architecture of single neuron of an ANN

By the associated weight (w, w , ..., w) and then sums up all the results i.e. computes a weighted sum of the input signals as:

$$X=\sum_{i=1}^{n} x_i w_i \qquad \ldots\ldots\ldots\ldots(1.1)$$

Then, the result is then passed through a non-linear function (f) called activation function. An activation function is the function that describes the output behavior of a neuron, and has a threshold value ′θ′. Then the result is compared with a threshold value which gives the output as either ‘0′ or ‘1′. If the weighted sum is less than ′θ′ then neuron output is ’1’ otherwise 0. In general, the neuron uses step function (1.2) as activation functions.

$$Y = \begin{cases} +1, if X \geq \theta \\ 0, if X < \theta \end{cases} \qquad \ldots\ldots\ldots (1.2)$$

Algorithm

1. Arithmetic Logic Units

```
import tensor flow as tf
def simple ALU(operand1, operand2, operation):
    if operation == "add":
        return tf.add(operand1, operand2)
    elif operation == "subtract":
        return tf.subtract(operand1, operand2)
    elif operation == "and":
        return          tf.logical_and(operand1,
operand2)
    elif operation == "equal":
        return tf.equal(operand1, operand2)
    else:
        raise            ValueError("Unsupported
operation")

# Example usage
a = tf.constant([1, 2, 3], dtype=tf.int32)
b = tf.constant([4, 1, 3], dtype=tf.int32)
c    =    tf.constant([True,    False,    True],
dtype=tf.bool)
d    =    tf.constant([False,    False,    True],
dtype=tf.bool)

sum result = simple alu(a, b, "add")
equal result = simple alu(c, d, "equal")

print("Sum:", sum result.numpy())
print("Equality:", equal result.numpy())
```

Algorithm

2. OR Gate using Tensor Flow:

In OR Gate, If one or both the inputs are High (True) then the output will be high (True) else low (False).

OR Gate

A	B	Z = A + B
False	False	False
False	True	True
True	False	True
True	True	True

```
import tensor flow as tf
# Define the input tensors for the OR gate
A = tf.constant([False, False, True, True],
dtype=tf.bool)
B = tf.constant([False, True, False, True],
dtype=tf.bool)

# Implement the OR gate using the Tensor Flow
logical or function
Z = tf.math.logical or (A, B)

# Print the output tensor
```

Journal of Artificial Intelligence and Emerging Technologies
Volume 3, Issue 4, pp 42-48, April-2026
ISSN (online): 3049-1967
DOI: https://doi.org/10.47001/JAIET/2026.304006
Available Online at https://www.jaiet.com

```
Print (Z)
```

Algorithm

3. AND Gate using Tensor Flow:

In AND Gate, If both the inputs are High (True) then the output will be high (True) else low (False).

AND Gate

A	B	Z = A.B
False	False	False
False	True	False
True	False	False
True	True	True

```
import tensor flow as tf
# Define the input tensors for the AND gate
A = tf.constant([False, False, True, True],
dtype=tf.bool)
B = tf.constant([False, True, False, True],
dtype=tf.bool)

# Implement the AND gate using the Tensor Flow
logical and function
Z = tf.math.logical_and(A, B)

# Print the output tensor
Print (Z
```

Algorithm

4. NAND Gate using TensorFlow:

```
import tensor flow as tf
import numpy as np

# 1. Define the training data (NAND gate truth
table)
A = np.array([False, False, True, True],
dtype=np.float32)
B = np.array([False, True, False, True],
dtype=np.float32)

# Implement the NAND gate using TensorFlow
logical_and and logical_not functions
Z = tf.math.logical_not(tf.math.logical_and(A,
B))

# Print the output tensor
print(Z)
```

Algorithm

5. NOT Gate using Tensor Flow:

In NOT Gate, The output will be opposite to the input value. i. e If the input is igh then the output will be Low and vice versa.

NOT Gate

A	Z = NOT A
False	True
Rue	False

```
import tensor flow as tf

# Define the input tensors for the NOT gate
A = tf.constant([False, True], dtype=tf.bool)

# Implement the NOT gate using the Tensor Flow
logical not function
Z = tf.math.logical not(A)

# Print the output tensor
Print (Z)
```

Algorithm

6. XNOR Gate using Tensor Flow:

In XNOR Gate, The output will be high only if both the inputs will be the same i. e either High or Low.

Similar to the NOR gate, Tensor flow does not have a function for the XNOR gate. Thus, we will combine logical XOR and logical NOT functions to construct the XNOR gate.

XNOR Gate

A	B	Z = A XOR B	Y = NOT Z	XNOR GATE = Y
False	False	False	True	True
False	True	True	False	False
True	False	True	False	False
True	True	False	True	True

```
import tensor flow as tf

# Define the input tensors for the XOR gate
A = tf.constant([False, False, True, True],
dtype=tf.bool)
B = tf.constant([False, True, False, True],
dtype=tf.bool)
```

Journal of Artificial Intelligence and Emerging Technologies
Volume 3, Issue 4, pp 42-48, April-2026
ISSN (online): 3049-1967
DOI: https://doi.org/10.47001/JAIET/2026.304006
Available Online at https://www.jaiet.com

```
# Implement the XNOR gate using the
# Tensor Flow logical xor function and logical
_not
# First implement OR Gate between A and B
Z = tf.math.logical xor(A, B)
# Now implement NOT Gate to output of XOR GATE
i.e 'Z'

Y = tf.math.logical not(Z)
# Print the output tensor
Print (Y)
```

Algorithm

7. XOR Gate using Tensor Flow:

In XOR Gate, If any one of the inputs is high then the output will be high and if both the inputs are high then the output will be low.

XOR Gate

X	Y	Z = A XOR B
False	False	False
False	True	True
True	False	True
True	True	False

```
import tensor flow as tf

# Define the input tensors for the XOR gate
A = tf.constant([False, False, True, True],
dtype=tf.bool)
B = tf.constant([False, True, False, True],
dtype=tf.bool)

# Implement the XOR gate using the Tensor Flow
logical xor function
Z = tf.math.logical xor (A, B)

# Print the output tensor
Print (Z)
```

IV. RESULTS

Neural Network Predictions:

1. Input tensors for the ALU gate:
Tf . Tensor flow [false True True]

2. Input tensors for the OR gate:
Tf . Tensor flow [false True True True]

3. Input tensor for the AND gate:
Tf . Tensor flow [False False False True]

4. Import tensor for the NAND gate:
Tf . Tensor flow [True True True False]

5. Import tensor for the NOT gate:
Tf . Tensor flow [True False]

6. Import tensor for the XNOR gate:
Tf . Tensor flow [True False False True]

7. Import tensor for the XOR gate:
Tf . Tensor flow [False True True False]

V. CONCLUSIONS

We demonstrated the effectiveness of neural arithmetic on several benchmark tasks and showed that they outperform traditional arithmetic in terms of accuracy and robustness. In the context of a neural arithmetic, an epoch and loss are two important concepts that are used to train and evaluate the performance of the model. An epoch is a single pass through the entire training dataset. In other words, it is one iteration of the training process where the model sees each example in the training dataset once. The number of epochs is a hyper parameter that needs to be tuned. Typically, the model is trained for multiple epochs until the loss converges or the performance on the validation set starts to degrade. The loss, also known as the cost function or objective function, is a measure of how well the model is doing on the training dataset. The test accuracy is 100 percent. Future work includes exploring the use of neural arithmetic in other applications, such as communication systems and data processing. We also plan to investigate the use of neural arithmetic in real-world scenarios.

REFERENCES

[1] Stanislaw Antol, Aishwarya Agrawal, Jiasen Lu, Margaret Mitchell, Dhruv Batra,C Lawrence Zitnick, and Devi Parikh. VQA: Visual question answering. *In Proceedings of the IEEE International Conference on Computer Vision*, pages 2425–2433, 2015.

[2] Carlos Arteta, Victor Lempitsky, J Alison Noble, and Andrew Zisserman. Interactive object counting. *In European Conference on Computer Vision*, pages 504–518, 2014.

[3] Steven L Brunton, Joshua L Proctor, and J Nathan Kutz. Discovering governing equations from data by sparse identification of nonlinear dynamical systems.

Journal of Artificial Intelligence and Emerging Technologies
Volume 3, Issue 4, pp 42-48, April-2026
ISSN (online): 3049-1967
DOI: https://doi.org/10.47001/JAIET/2026.304006
Available Online at https://www.jaiet.com

Proceedings of the National Academy of Sciences, 113(15):3932–3937, 2016.

[4] Antoni B Chan, Zhang-Sheng John Liang, and Nuno Vasconcelos. Privacy preserving crowd monitoring: Counting people without people models or tracking. *In Proc. CVPR*, pages 1–7 IEEE, 2008.

[5] Stanislas Dehaene. The Number Sense: How the Mind Creates Mathematics. *Oxford University Press*, 2011.

[6] Jerry A. Fodor and Zenon W. Pylyshyn. Connectionism and cognitive architecture: a critical analysis. *Cognition*, 28(1–2):3–71, 1988.

[7] C. Randy Gallistel. Finding numbers in the brain. *Philosophical Transactions of the Royal Society B*, 373, 2017.

[8] Rochel Gelman and C. Randy Gallistel. The child's understanding of number. *Harvard*, 1978.

[9] Felix A Gers and E Schmidhuber. LSTM recurrent networks learn simple context-free and context-sensitive languages. *IEEE Transactions on Neural Networks*, 12(6):1333–1340, 2001.

[10] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. *In Acoustics, speech and signal processing (ICASSP), 2013 IEEE international conference on*, pages 6645–6649. IEEE, 2013.

[11] Alex Graves, Greg Wayne, and Ivo Danihelka. Neural Turing machines. *CoRR*, abs/1410.5401, 2014. URL http://arxiv.org/abs/1410.5401.

[12] Alex Graves, Greg Wayne, Malcolm Reynolds, Tim Harley, Ivo Danihelka, Agnieszka Grabska- Barwi´nska, Sergio Gómez Colmenarejo, Edward Grefenstette, Tiago Ramalho, John Agapiou, et al. Hybrid computing using a neural network with dynamic external memory. *Nature*, 538 (7626):471, 2016.

[13] Edward Grefenstette, Karl Moritz Hermann, Mustafa Suleyman, and Phil Blunsom. Learning to transduce with unbounded memory. *In Proc. NIPS*, 2015.

[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. *CoRR*, abs/1603.05027, 2016. URL http://arxiv.org/abs/1603.05027.

[15] Gao Huang, Zhuang Liu, and Kilian Q. Weinberger. Densely connected convolutional networks. *CoRR*, abs/1608.06993, 2016. URL http://arxiv.org/abs/1608.06993.

[16] Rafal Jozefowicz, Oriol Vinyals, Mike Schuster, Noam Shazeer, and Yonghui Wu. *Exploring the limits of language modeling*. arXiv preprint arXiv:1602.02410, 2016.

[17] Alex Krizhevsky and Geoff Hinton. Convolutional deep belief networks on CIFAR-10. *Unpub- lished manuscript*, 2010.

[18] Ankit Kumar, Ozan Irsoy, Peter Ondruska, Mohit Iyyer, James Bradbury, Ishaan Gulrajani, Victor Zhong, Romain Paulus, and Richard Socher. Ask me anything: Dynamic memory networks for natural language processing. *In Proc. ICML*, pages 1378–1387, 2016.

[19] Brendan Lake and Marco Baroni. Generalization without systematicity: On the compositional skills of sequence-to-sequence recurrent networks. *In Proc. ICML*, 2018.

[20] Gary F. Marcus. The Algebraic Mind: Integrating Connectionism and Cognitive Science. *MIT Press*, 2003.

[21] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lilli- crap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. *In Proc. ICML*, pages 1928–1937, 2016.

[22] Manuela Piazza, Véronique Izard, Philippe Pinel, Denis Le Bihan, and Stanislas Dehaene. Tuning curves for approximate numerosity in the human intraparietal sulcus. *Neuron*, 44: 547–555, 2004.

[23] Scott E. Reed and Nando de Freitas. Neural programmer-interpreters. *In Proc. ICLR*, 2016.

[24] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533, 1986.

[25] Santi Seguí, Oriol Pujol, and Jordi Vitrià. Learning to count with deep object features. *CoRR*, abs/1505.08082, 2015. URL http://arxiv.org/abs/1505.08082.

Journal of Artificial Intelligence and Emerging Technologies
Volume 3, Issue 4, pp 42-48, April-2026
ISSN (online): 3049-1967
DOI: https://doi.org/10.47001/JAIET/2026.304006
Available Online at https://www.jaiet.com

Citation of this Article:

Faiqua Rizwan, Priyanjala Yadav, Shikha Yadav, & Mohd. Imran Aziz. (2026). AI-Driven Arithmetic Logic Units. *Journal of Artificial Intelligence and Emerging Technologies (JAIET).* 3(4), 42-48. Article DOI: https://doi.org/10.47001/JAIET/2026.304006

***** End of the Article *****