frontiers in Artificial Intelligence

ORIGINAL RESEARCH
published: 29 September 2020
doi: 10.3389/frai.2020.00071

iNALU: Improved Neural Arithmetic Logic Unit

Daniel Schlör[1]*, *Markus Ring*[2] *and Andreas Hotho*[1]

[1] *Data Science Chair, Institute of Computer Science, University of Wuerzburg, Würzburg, Germany,* [2] *Department of Electrical Engineering and Computer Science, University of Applied Sciences and Arts Coburg, Coburg, Germany*

Neural networks have to capture mathematical relationships in order to learn various tasks. They approximate these relations implicitly and therefore often do not generalize well. The recently proposed Neural Arithmetic Logic Unit (NALU) is a novel neural architecture which is able to explicitly represent the mathematical relationships by the units of the network to learn operations such as summation, subtraction or multiplication. Although NALUs have been shown to perform well on various downstream tasks, an in-depth analysis reveals practical shortcomings by design, such as the inability to multiply or divide negative input values or training stability issues for deeper networks. We address these issues and propose an improved model architecture. We evaluate our model empirically in various settings from learning basic arithmetic operations to more complex functions. Our experiments indicate that our model solves stability issues and outperforms the original NALU model in means of arithmetic precision and convergence.

Keywords: neural networks, machine learning, arithmetic calculations, neural architecture, experimental evaluation

OPEN ACCESS

Edited by:
Devendra Singh Dhami,
The University of Texas at Dallas,
United States

Reviewed by:
Mayukh Das,
Samsung, India
Alejandro Molina,
Darmstadt University of Technology,
Germany

****Correspondence:***
Daniel Schlör
schloer@informatik.uni-wuerzburg.de

Specialty section:
This article was submitted to Machine Learning and Artificial Intelligence,
a section of the journal
Frontiers in Artificial Intelligence

Received: *01 April 2020*
Accepted: *05 August 2020*
Published: *29 September 2020*

Citation:
Schlör D, Ring M and Hotho A (2020) iNALU: Improved Neural Arithmetic Logic Unit. Front. Artif. Intell. 3:71.
doi: 10.3389/frai.2020.00071

1. INTRODUCTION

Neural networks have achieved great success in various machine learning application areas. Different network structures have proven to be suitable for different tasks. For instance, convolutional neural networks are well-suited for image processing while recurrent neural networks are well-suited for handling sequential data. However, neural networks also face challenges like processing categorical values or calculating specific mathematical operations.

The presence of mathematical relationships between features is a well-known fact in many financial tasks (Bolton and Hand, 2002; Lopez-Rojas et al., 2016). Other examples can be found in the intrusion detection domain. For example, some intrusion detection methods count the number of certain events (Garcia et al., 2014) or consider some restrictions such as network packets having a minimum and maximum number of transmitted bytes (Ring et al., 2019). A model which is able to capture these relationships explicitly in an automated way is therefore very desirable and can be incorporated in various machine learning tasks.

Problem

While neural networks are successfully applied in complex machine learning tasks, single neurons often have problems with the calculation of basic mathematical operations (Trask et al., 2018). This fact can be explained by inspecting the structure of neurons in detail. The output of a neuron i is the weighted sum of all input signals, an optional bias b and an activation function:

$$output_i = act\left((\sum_{j=1}^{n} x_j \cdot w_j) + b_i \right) \tag{1}$$

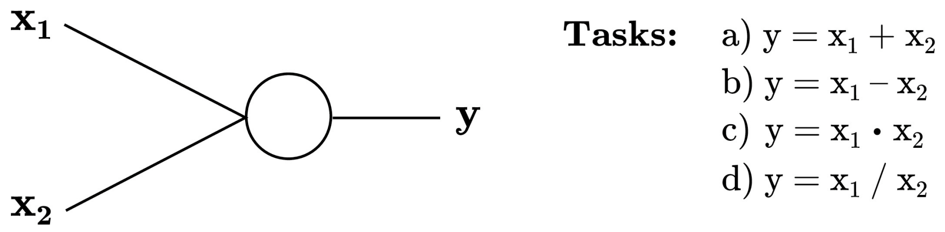

FIGURE 1 | Standard mathematical tasks.

The neuron i in Equation (1) receives n input signals x_j which are multiplied by the weights w_j. The parameter b_i represents an optional bias and $act(\cdot)$ is an arbitrary activation function like the identity for a linear or sigmoid for a non-linear neuron. This allows neurons to assign different weights to different input features. Further, linear neurons are able to add (or subtract) different inputs by setting their corresponding weights to 1 (or -1), see tasks (a) and (b) in **Figure 1**. However, activation functions, weights and bias allow neurons only to approximate the result of multiplications and divisions in their training range, since the output is the weighted sum of all inputs. Consequently, they can't solve multiplication and division tasks for values outside the training range [see tasks (c) and (d) in **Figure 1**].

Trask et al. (2018) show empirically that artificial neurons have especially difficulties with extrapolation of mathematical operations and present the *Neural Arithmetic Logic Units* (NALU) to address this problem. However, the NALU is only able to calculate non-negative results for multiplication and division by design. Madsen and Rosenberg Johansen (2019) further show that the NALU is not able to learn division reliably and often fails to converge to the desired weights.

Objective

Inspired by the NALU, we want to improve the architecture to address the above mentioned problems. Our focus lies on processing negative values and improving extrapolation by forcing the internal weights to intended values.

Contribution

In this paper, we propose iNALU as improvement of the NALU architecture (Trask et al., 2018). Our proposed architecture improves the stability, enables the network to calculate with negative and positive inputs and improves the precision of arithmetic tasks in general. Therefore we change several technical aspects of the original NALU. To be precise, we add another path to allow multiplication and division with mixed-signed inputs. Further, we propose an input independent implementation of the gate, switching between the summative and multiplicative path. Based on empirical observations, we add regularization to the training procedure to prevent approximation of the results due to unwanted combination of mathematical operations. Then, a maximum function for the multiplicative path is introduced to avoid too large values (infinity) for deep networks with several hidden layers and many neurons. We experimentally evaluate the improved architecture in various settings: Minimal arithmetic tasks, one-layer calculations where among others the relevant inputs have to be recognized and simple function learning where a combination between operations has to be learned in two layers.

Our main contributions are the improvement of the extrapolation results of the NALU and the mixed-signed multiplication with negative values as result. The iNALU code is available on github[1].

Structure

The paper is structured as follows: The next section describes related work. Section 3 explains the NALU and our improved model iNALU in more detail. Experiments are presented in section 4 and the results are discussed in section 5. Finally, section 6 concludes the paper.

2. RELATED WORK

This section reviews related work on processing mathematical operations using neural networks.

Kaiser and Sutskever (2016) present Neural GPU, a neural network architecture which is able to solve algorithmic tasks. The architecture of Neural GPU is based on a type of convolutional gated recurrent units (CGRU). The authors show that their approach is able to learn long binary summations and multiplications and that their approach generalizes well for longer numbers. However, in the experimental evaluation, the input to the network is limited to four symbols. Freivalds and Liepins (2018) propose an improvement for the Neural GPU which speeds up the training time and provides better generalization. Similarly, Kalchbrenner et al. (2015) propose Grid Long Short-Term Memory, a network of LSTM cells which is able to add 15-digit integer numbers. These three approaches from Kaiser and Sutskever (2016), Freivalds and Liepins (2018) and Kalchbrenner et al. (2015) process sequential data and are able to learn simple algorithmic tasks.

Another work in this area is proposed by Chen et al. (2018). The authors use reinforcement learning to solve mathematical operations such as summation, subtraction, multiplication or division. However, compared to our setting, Chen et al. provide the mathematical operation as an additional input to their network.

The most similar work to ours is from Trask et al. (2018). The authors propose the neural arithmetic logic unit which is

[1] https://github.com/daschloer/inalu

able to perform mathematical operations. They show in their experimental evaluation that their model generalizes better than traditional neurons for extrapolation tasks. However, the NALU has some limitations which we discuss in section 3.2.

Other works with small intersections are presented by Zaremba and Sutskever (2014) as well as by Reed and De Freitas (2015). Both use Recurrent Neural Networks to execute small code snippets which contain the summation of digits. Counting the number of specific objects in images can also be seen in the wider scope of related work. In this context, works by Xie et al. (2018) and Zhang et al. (2015) involve counting the number of microscopy cells respectively crowd counting.

3. IMPROVED NEURAL ARITHMETIC LOGIC UNIT

In this chapter, we first describe the Neural Arithmetic Logic Unit and discuss properties and challenges. We then introduce iNALU, a new model variant, to address these challenges.

3.1. Neural Arithmetic Logic Unit

The NALU as proposed by Trask et al. (2018) consists of a multiplicative and a summative path, which can be seen as a linear layer with a weight matrix constrained to $[-1, 1]$. The weights $\mathbf{W}$ are constructed as point-wise product between a matrix $\hat{\mathbf{W}}$ with tanh activations and a matrix $\hat{\mathbf{M}}$ with sigmoid (σ) activations.

$$\mathbf{W} = \tanh(\hat{\mathbf{W}}) \odot \sigma(\hat{\mathbf{M}}) \tag{2}$$

By matrix multiplication of inputs $\mathbf{x}$ and weights $\mathbf{W}$, output values stay within the magnitude of the input values (since $-1 \leq \mathbf{W}_{i,j} \leq 1$) and result in the summation for values of $\mathbf{W}_{i,j} = 1$ and subtraction for values of $\mathbf{W}_{i,j} = -1$. By balancing the weights between -1, 0, and 1 any function composed of adding, subtracting and ignoring inputs can be learned. This summative path $\mathbf{a}$ is defined in Equation (3).

$$\mathbf{a} = \mathbf{x}\mathbf{W} \tag{3}$$

To multiply or divide, this calculation is performed in log-space (see Equation 4). The NALU encounters the problem of calculating $\log(x)$ for $x \leq 0$ by restricting the calculation to absolute input values and adding a small constant value ϵ.

$$\mathbf{m} = \exp\left(\log(|\mathbf{x}| + \epsilon)\mathbf{W}\right) \tag{4}$$

A gate is used to decide between the summative and the multiplicative path depending on the input vector.

$$g = \sigma(\mathbf{x}\mathbf{G}) \tag{5}$$

Since the gate weights $\mathbf{G}$ are multiplied with the inputs $\mathbf{x}$, each gate dimension maps to an input dimension and contains the corresponding weight to which the input shall contribute to the decision between both arithmetic paths.

The output is obtained by adding the gated summative (see Equation 3) and multiplicative (see Equation 4) paths.

$$\texttt{NALU}_o\colon \mathbf{y}_{\text{nalu}} = g \cdot \mathbf{a} + (1 - g) \cdot \mathbf{m} \tag{6}$$

The NALU model can finally be implemented in two ways. One can either use a weight vector $\mathbf{G}$ and a scalar gate g or a weight matrix $\mathbf{G}$ and a gate vector $\mathbf{g}$. Tasks for which the selection of the operation is different for each output or for which it is depending on input values might benefit from the gate matrix. However, this introduces additional parameters which for many tasks are unnecessary. In our experiments we use both, vector based NALU and a NALU with matrix based gating for comparison which we refer to as NALU (v) respectively NALU (m).

However, some of these design decisions for the NALU result in challenges we want to address in the following section.

3.2. Challenges

3.2.1. Exploding Intermediate Results

In our experiments, we observe that training often fails because of exploding intermediate results especially when stacking NALUs to deeper networks and having many input and output variables. For example, consider a model consisting of four NALU layers with four input and output neurons each and a simple summation task. Assuming the same magnitude for all input dimensions the first layer could (depending on the initialization) calculate x^4 for each output dimension whereas the following layer could calculate $(x^4)^4$ ultimately leading to x^{4l} for layer l. Therefore, the calculation can exceed the valid numeric range already in the forward pass ultimately causing the training to fail. For example in a network with three NALU layers in an MNIST classification downstream task, the NALU models failed after the first training steps (resulting in NaNs).

3.2.2. Multiplication/Division With Negative Result

The NALU by design isn't capable of multiplying or dividing values with a negative result. In the multiplicative path, the input values are represented by their absolute value to guarantee a real-valued calculation in log-space. Therefore, learning multiplication for mixed signed data with a result $y < 0$ fails. Since the NALU is expected to learn either multiplication / division or summation / subtraction in each layer, $\text{sign}(y)$ is in the multiplicative case clearly determined by the number of negative multiplicands being even or odd. Since input dimensions can be deactivated for $\mathbf{W}_{i,j} = 0$, the sign can't be inferred counting negative input variables. In the next section, we propose a method taking deactivated input dimensions into account to correct the sign of the multiplicative path.

3.2.3. Mixed Sign Gating

Despite the summative path is capable of dealing with mixed input signs, the construction of the gating mechanism leads to problems. If input values are constantly positive or constantly negative, Equation 5 leads to the desired gating behavior. However, if the input values mix negative and positive values, σ and thus the gate is dependent of the sign since $\mathbf{G}$ can't fit the designated gate state systematically correctly.

3.2.4. Initialization Sensitivity

We observed that the NALU architecture is very prone to non-optimal initializations, which can lead to vanishing gradients or optimization into undesired local optima. Finding the optimal initialization in general is difficult since it depends on the task and the input distribution, which in a real world scenario is both unknown.

3.2.5. Leaky Gates

Another challenge we observe are variables, not tied near to their boundaries. Generally in the NALU design the variables $\mathbf{W}$ and g are intended to reach their boundaries of $[-1, 1]$ and $[0, 1]$ for maximum precision. However, during training and for interpolation, an approximation of the intended calculation having gates trained to $g = 0.5$, for example with a specific configuration of $\mathbf{W}$ represents a local optimum. For extrapolation such a model fails by large margin. We suggest regularizing the trained variables to avoid this behavior.

3.3. Improvements

This section describes the improvements we incorporate in our iNALU model to address the aforementioned challenges. **Figure 2** summarizes the complete model architecture. In the following, we discuss each improvement and extension in detail.

3.3.1. Independent Weights

The summative and the multiplicative paths share their weights $\hat{\mathbf{W}}$ and $\hat{\mathbf{M}}$ in the NALU model. We propose using separate weights for each path for two reasons: First, the model can optimize $\mathcal{W}$ for the multiplicative and summative path without interfering the other path. For example, in a setting with inputs $a, b < -1$ with the operation $a \times b$, the result would be a positive number greater than 1 and the optimal parameter setting would be $\mathbf{W}_a = \mathbf{W}_b = 1$ and $g = 0$. However, the only way for the summative path (see Equation 3) to generate positive results is to force the weights $\mathbf{W}_a$ and $\mathbf{W}_b$ toward -1. In this case, the summative and multiplicative path force the weights into opposite directions. With separate weights, the model can learn optimal weights for both paths and select the correct path using the gate. Second, consider the multiplicative path yields huge results whereas the summative path represents the correct solution but yields relatively small results. In that case, the multiplicative path influences the results even if the sigmoid gate is almost closed. For example in a setting with inputs $a, b, c > 0$ with the desired result $a+b$, the summative path yields the correct solution and the optimal weight setting is $\mathbf{W}_a = \mathbf{W}_b = 1, \mathbf{W}_c = 0$ and $g = 1$. In that case, $\mathbf{W}$ may contain very small weights to omit the input c. However, small negative weights for $\mathbf{W}_c$ (e.g., -10^{-5}) leads to the situation, that the multiplicative path divides the inputs a and b by values near to 0 which results in large numbers. Consequently, the multiplicative path influences the results even if the gate (see Equation 5) is almost closed. In this case, the model with independent weights can optimize $\mathbf{W}_\mathrm{m}$ to smaller values to mitigate influence caused by the leaky gate. Our modifications are summarized in the following equations:

$$\mathbf{W}_\mathrm{a} = \tanh(\hat{\mathbf{W}}_a) \odot \sigma(\hat{\mathbf{M}}_a) \tag{7}$$

$$\mathbf{W}_\mathrm{m} = \tanh(\hat{\mathbf{W}}_m) \odot \sigma(\hat{\mathbf{M}}_m) \tag{8}$$

$$\mathbf{a} = \mathbf{x}\mathbf{W}_\mathrm{a} \tag{9}$$

$$\mathbf{m} = \exp\left(\log(|\mathbf{x}| + \epsilon)\mathbf{W}_\mathrm{m}\right) \tag{10}$$

3.3.2. Weight and Gradient Clipping

To address the challenge of exploding intermediate results in a multi-layer setting, we improve the model by clipping exploding weights in the back-transformation from log-space (see Equation 11) and avoid calculating imprecisely by incorporating ϵ and ω only if $\mathbf{x}$ values caused exploding intermediate results.

$$\mathbf{m} = \exp\Big(\min\big(\log\big(\max(|\mathbf{x}|, \epsilon)\big)\,\mathbf{W}_\mathrm{m}, \omega\big)\Big) \tag{11}$$

This kind of weight clipping is a simple practical solution to improve the stability of deep iNALU networks which has for example been successfully employed in Wasserstein Generative Adversarial Networks (Arjovsky et al., 2017). The original NALU architecture didn't address this problem causing practical stability issues. To validate this, we incorporated three NALU layers in a MNIST classification downstream task. Our proposed clipping mechanism resulted in successful training solving the task very well[2], whereas the original NALU fails producing NaNs.

This shows the effectiveness of our proposed improvement, albeit more sophisticated solutions might be an interesting topic of future work to avoid vanishing gradients for clipped neurons. Further, we apply gradient clipping to avoid stability problems due to large gradients, which can for example occur when input values are near to zero. We set ϵ to 10^{-7} and ω to 20.

3.3.3. Sign Correction

The NALU cell by design isn't capable of multiplying or dividing values with a negative result. Therefore, NALU fails calculating multiplication of mixed signed data. Considering the sign within the log-space transformation is not trivial since $\log(x)$ is not defined for $x < 0$ in $\mathbb{R}$. Instead inferring the correct sign *post-hoc* is a more feasible solution, which follows the human intuition of multiplying or dividing numbers with mixed signs. However, multiplying over the $\mathrm{sign}\,(\mathbf{x})$ vector doesn't provide a universal solution, since some input dimensions may be deactivated ($\mathbf{W}_{i,j} = 0$). We propose a solution by taking the sign of only relevant input values into account (i.e., all $\mathbf{W}_{i,j} \neq 0$).

$$\mathbf{msm_1} = \mathrm{sign}(\mathbf{x}) \odot |\mathbf{W}_\mathrm{m}| \tag{12}$$

$$\mathbf{msm_2} = 1 - |\mathbf{W}_\mathrm{m}| \tag{13}$$

$$\mathbf{msm} = \mathbf{msm_1} + \mathbf{msm_2} \tag{14}$$

$$\mathbf{msv} = \prod_i \mathbf{msm}_{ij} \tag{15}$$

$$\texttt{iNALU}_s\text{: } \mathbf{y}_\mathrm{nalu} = g \cdot \mathbf{a} + (1-g) \cdot \mathbf{m} \odot \mathbf{msv} \tag{16}$$

The sign correction is independent of the operation in the multiplicative path and has to be applied for multiplication and

[2] With an accuracy of 0.94 after 64000 steps.

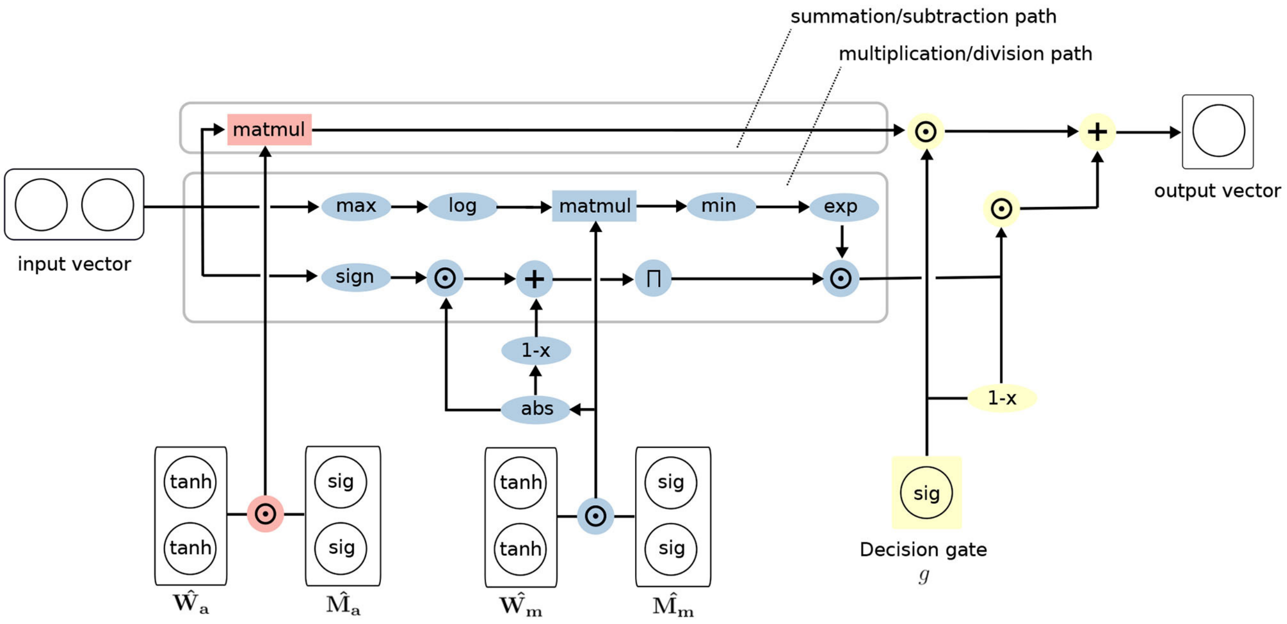
summation/subtraction path
multiplication/division path
matmul
input vector
max
log
matmul
min
exp
sign
⊙
+
∏
⊙
1-x
abs
output vector
⊙
+
⊙
1-x
tanh
tanh
⊙
sig
sig
tanh
tanh
⊙
sig
sig
sig
Decision gate
g
$\hat{\mathbf{W}}_\mathbf{a}$
$\hat{\mathbf{M}}_\mathbf{a}$
$\hat{\mathbf{W}}_\mathbf{m}$
$\hat{\mathbf{M}}_\mathbf{m}$

FIGURE 2 | Architecture of the improved Neural Arithmetic Logic Unit (iNALU).

division. Therefore, we use the absolute value of the weight matrix W_m to identify relevant and irrelevant input values. First, the sign function is applied to the input vector $\mathbf{x}$ which then is multiplied element by element with the absolute value of the weight matrix W_m, which leads to +1 for positive relevant inputs, -1 for negative relevant inputs and 0 for irrelevant inputs (see Equation 12). The multiplication of all row elements (input dimensions) per column of $\mathbf{msm_1}$ leads to 0, if any input dimension is irrelevant ($\mathbf{W}_{i,j} = 0$). To prevent this, we represent all irrelevant inputs as +1, since +1 does not influence the result of a multiplication. We achieve this by introducing a second matrix $\mathbf{msm_2}$ (see Equation 13) which is +1 for irrelevant inputs and 0 for relevant inputs and add $\mathbf{msm_1}$ and $\mathbf{msm_2}$. Finally, we infer the sign vector containing the sign of the multiplicative path for each output dimension (see Equation 15) by multiplying over each column of $\mathbf{msm}$. This sign represents the correct solution, if $\mathbf{W}$ is discrete i.e., $\mathbf{W}_{i,j} \in \{-1, 0, 1\}$. Discrete weights are a desired property (Trask et al., 2018) to achieve generalization and interpretability and ensure that $\mathbf{msm}$ is also discrete, i.e., $\mathbf{msm}_{i,j} \in \{-1, 1\}$. By introducing regularization (see section 3.3.4), we force the model to find discrete weights $\mathbf{W}$. Implementing both improvements enables the model to calculate inputs with mixed signs for the multiplicative path correctly.

3.3.4. Regularization

In general, $\mathbf{W}$ and g having discrete values is often crucial for a model to generalize and learn a calculation correctly instead of approximating the solution. This becomes even more important for the sign corrected multiplication. We therefore propose regularizing the weights such that $\hat{\mathbf{W}}$, $\hat{\mathbf{M}}$, and $\mathbf{G}$ don't contain values near zero by introducing a piecewise linear regularization term (see Equation 17) which adds to the loss until the weight has reached a discretization threshold t. We found $t = 20$ suitable since $\sigma(-20) < 10^{-9}$ and $1 - \tanh(20) < 10^{-17}$.

$$\mathcal{L}_{\text{reg}}(w) = \frac{1}{t}\max(\min(-w, w) + t, 0) \tag{17}$$

Consider for example weights $-t < \hat{w}, \hat{m} < t$, e.g., $\hat{w} = \hat{m} = 0.3$. Applying Equation 2 to these weights results in $w = \tanh(\hat{w}) \cdot \sigma(\hat{m}) \approx 0.167$. This means that the corresponding input is scaled down for the calculation of the additive or multiplicative path. Although mixing scaled inputs might result in suitable approximations of the underlying training data, usually such solutions fail to generalize the function and thus to extrapolate. By incorporating regularization loss, the model has a small gradient forcing the weights $\hat{w}$ and $\hat{m}$ toward $-t$ respectively t. Therefore the model is penalized for (local) approximations with values $-t < \hat{w}, \hat{m} < t$ pushing $\tanh(\hat{w})$ toward 1 or -1 and $\sigma(\hat{m})$ toward 0 or 1.

Note that the regularization can cause gradient-directions contradicting the gradient-direction of the loss without regularization depending on the initialization. We try to mitigate this problem by incorporating the regularization only after several training steps, when the loss is below a threshold (see section 4 for more details).

Further, regularization is especially useful to improve extrapolation performance. For example, we evaluate regularization in the Simple Function Learning Task (see section 4.7) setup for a summation task (i.e., an overdetermined task where an optimal and generalizing solution can be found even for $-1 < \mathbf{W}_{i,j} < 1$). We obtained after 10 epochs without regularization an interpolation loss of $5.95 \cdot 10^{-4}$ and an extrapolation loss of $4.46 \cdot 10^{11}$. The model has found a suitable approximation for the training range but failed to generalize. Introducing regularization after the 10th epoch and evaluating after 15 epochs we reach an interpolation loss of $2.2 \cdot 10^{-13}$ and an extrapolation loss of $2.2 \cdot 10^{-11}$, whereas without regularization we just improve the interpolation loss ($8.30 \cdot 10^{-5}$) and the extrapolation loss even impairs ($8.76 \cdot 10^{14}$).

3.3.5. Reinitialization

Since NALU doesn't recover well from local optima by its own (Madsen and Rosenberg Johansen, 2019), we suggest a reinitialization strategy. This strategy evaluates the loss for each m-th epoch and randomly reinitializes all weights if the loss did not improve for the last n steps and if the loss is greater than a predefined threshold.

3.3.6. Independent Gating

In the original NALU model the gate deciding between the multiplicative and the additive path is calculated by multiplying the input vector and the gate weight matrix **G** (see Equation 5). While this can be beneficial if the decision between operations is encoded in an op-code alike fashion, in many tasks, the decision which operation path to choose is not depending on the input values but instead fixed for the task, e.g., typical spreadsheet tasks like calculating the sum or product of different columns.

For this case we propose a model, where the scalar gate is replaced by a vector which is, contrarily to the original NALU model, independent from the input (see Equation 18). Thereby the gate weights are indirectly optimized through back-propagation during training of the network to represent the best-fitting operation, reminiscent of training bias in a linear layer.

$$\mathbf{g} = \sigma(\mathbf{G}) \tag{18}$$

For example consider a NALU network with one layer, the operation $+$ and the inputs $\mathbf{x_1} = (2, 2)$ and $\mathbf{x_2} = -\mathbf{x_1} = (-2, -2)$ resulting in the calculations $2 + 2 = 4$ and $-2 + (-2) = -4$. For the original NALU the function $y = \sigma(xG) \cdot a + (1 - \sigma(xG)) \cdot m$ has to be optimized, i.e., a G has to be found which holds $\sigma(xG) \to 1$ for all x to choose the correct operation. Both inputs $\mathbf{x_1}$ and $\mathbf{x_2}$ have to be calculated with the same operation $+$, thus for a suitable G, $\sigma(\mathbf{x_1}G) = \sigma(\mathbf{x_2}G) \Leftrightarrow \sigma(\mathbf{x_1}G) = \sigma(-\mathbf{x_1}G)$ must hold. Since $G = 0$ is the only solution leading to $\sigma(xG) = \sigma(0) = 0.5$, a valid solution satisfying both constraints doesn't exist. With independent gating, the iNALU can optimize $\sigma(G) \to 1$ up to an arbitrary precision and therefore learn the function correctly.

Additional choosing a vector over a scalar enables our model to select the operation for each output independently introducing the capability to calculate for example $y_1 = a + b$, $y_2 = a \cdot b$ for an input $x = (a, b)$ simultaneously.

4. EXPERIMENTS

4.1. Design of Experiments

In this section, we perform an experimental evaluation of the proposed iNALU model to analyze its basic abilities to solve mathematical tasks in comparison to the original NALU. Precisely, we compare two NALU models, NALU (v) with a gate vector **G**, NALU (m) with gate matrix **G** with two iNALU models, iNALU (sw) with shared weights between the additive and multiplicative path and iNALU (iw) with independent weights for each path. In this section, experiments of varying complexity are conducted to examine several research questions:

Experiment 1 examines the research question, how well each model performs in its minimal setup for different input distributions i.e., one layer with two input and one output neurons. We show that the iNALU outperforms the NALU and reaches very low error rates for almost all distributions.

In experiment 2 we evaluate how well the models perform on different magnitudes of input data. The results show that, the iNALU models can reach a high precision for data of different magnitude, albeit the precision for multiplication impairs with increasing magnitude of input data.

Experiment 3 examines the capability of each model to ignore input dimensions. We show that the iNALU is capable of learning to ignore input dimensions well, whereas the original NALU fails for most operations and distributions.

With experiment 4 we compare different initialization strategies. The parameter study shows that the initialization has a large impact on the stability of the network. We finally identify the most suitable parameter configuration for more complex tasks.

Finally experiment 5 examines the performance of NALU and iNALU models for a function learning task involving two arithmetic operations per function using architectures with two layers and 100 input dimensions. We show that the iNALU models outperform both NALU models by large margin and yield a very high precision for all operations except division.

4.2. Prerequisites

This section describes at first the general commonalities of all experiments.

4.2.1. Datasets

For all experiments, we evaluate on an interpolation task as well as an extrapolation task. For the interpolation task, the training and evaluation dataset are drawn from the same distribution. For the extrapolation task, the evaluation dataset is drawn from a distribution with a different value range in order to evaluate the ability to generalize. Each dataset contains $N = 64\,000$ samples.

4.2.2. Tasks

For our experiments we focus on mathematic operations since these are the building-blocks of more complex tasks. All tasks involve applying an operation $\diamond \in \{+, -, \times, \div\}$ to input and/or hidden variables a and b to calculate $y = a \diamond b$. Note that Trask et al. (2018) introduces additional operations such as identity, square and the square-root but since these operation are special cases of the basic operations, their learning performance is closely correlated with the performance on the basic operations and therefore omitted for the sake of clarity. The input variables for all experiments are sampled randomly from a distribution $\mathcal{P}$ with a parameterization λ, which are defined in the following sections in more detail. Note that for $\mathcal{P} = \mathcal{N}$ the normal distribution for our experiments is truncated to $\lambda = [a, b] =$

$[\mu - 3\sigma, \mu + 3\sigma]$ (containing $\approx$ 99,7% probability mass) to ensure that the extrapolation task is performed out of the test distribution range. For the exponential distribution ($\mathcal{P} = \mathcal{E}$) the extrapolation task involves no extrapolation in a literal sense but rather examines if generalization for different λ values can be achieved.

4.2.3. Evaluation

In contrast to Trask et al. (2018), we choose a different evaluation strategy: Trask et al. reported the error for each operation relatively in comparison to a random initialized network prior training. Since the performance of the untrained network is constantly bad, the relative performance reported this way can be used to decide how well each architecture performs rank-wise but it can't be used to infer, to which extend the calculated result differs from the expected result. Instead, we use a more intuitive approach for evaluation and report the mean squared error (MSE) between the calculated and the expected results over the complete evaluation datasets. For all experiments we report results for extrapolation, since this is the more difficult task.

$$\mathrm{MSE}(y^{\mathrm{pred}}, y^{\mathrm{real}}) := \frac{1}{N}\sum_{i}^{N}(y_i^{\mathrm{pred}} - y_i^{\mathrm{real}})^2 \tag{19}$$

The MSE comes along with another advantage. Combined with a predefined threshold, the MSE can be used to evaluate if the model reaches the necessary precision (Madsen and Rosenberg Johansen, 2019). If not stated otherwise we understand a MSE $\leq 10^{-4}$ as successful training.

We repeat each experiment ten times with different random seeds. This procedure examines if the performance is stable or how much it scatters randomly.

4.2.4. Training

We use the Adam optimizer (Kingma and Ba, 2015) in mini-batch training with a learning-rate of 0.001 and a batch size of 64. Training is done for 100 epochs using the MSE as loss. Clipping, regularization and random reinitialization as described in section 3.3 are implemented. Regularization is activated after 10 epochs whenever the training loss $\mathcal{L} < 1$. Reinitialization is applied each 10th epoch if the loss hasn't improved over $m = 10,000$ steps.

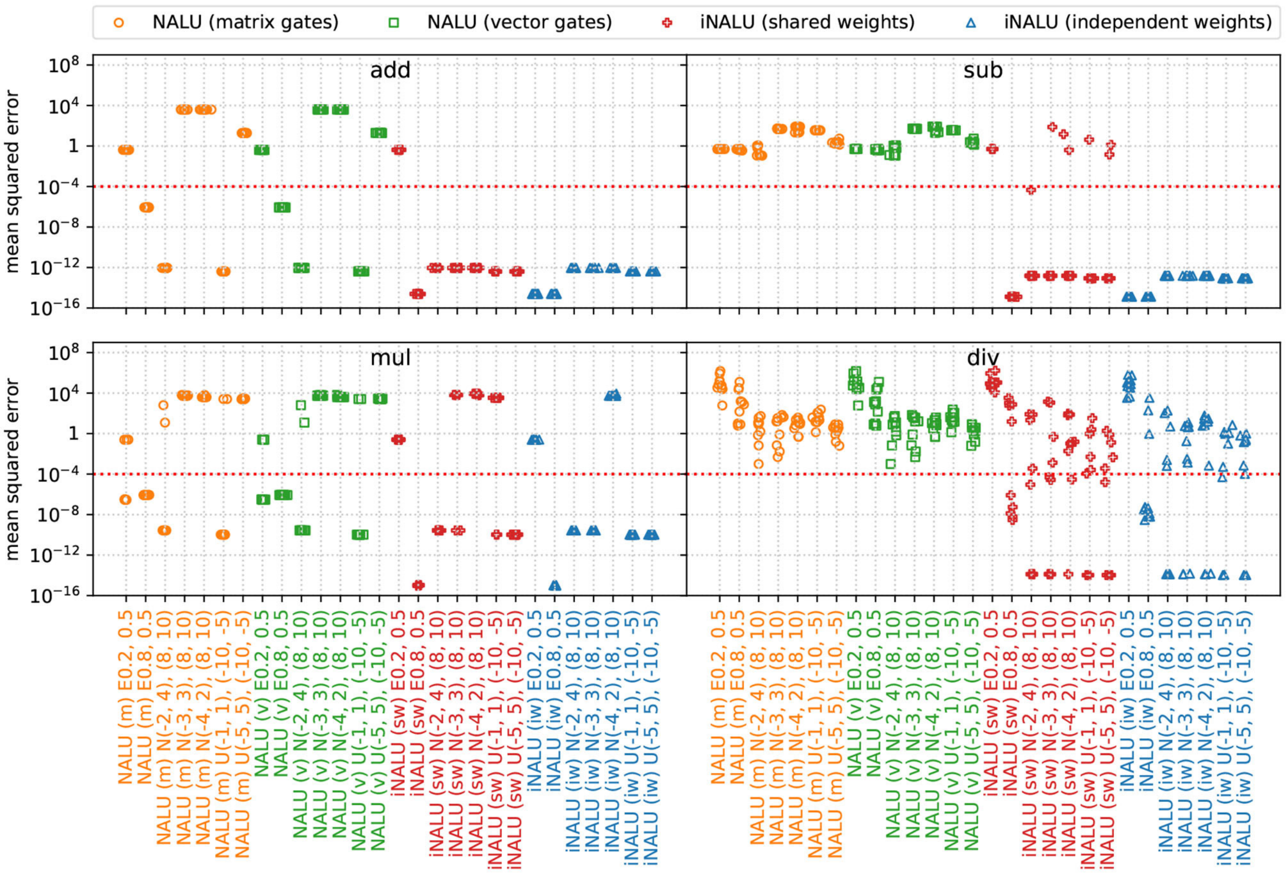

FIGURE 3 | MSE for various input distributions per operation over the extrapolation test dataset of experiment 1 (minimal arithmetic task). The original NALU is colored in orange and green, (m) stands for the matrix gating, and (v) for the vector gating version. Our iNALU models are depicted in red for the shared weights variant and blue for the version with independent weight matrices for the summative and multiplicative path. For truncated normal (N) as for uniform distributed data (U), the first parameter tuple represents the training data range, the second tuple represents the extrapolation range. For exponentially distributed data (E) the parameter λ is reported.

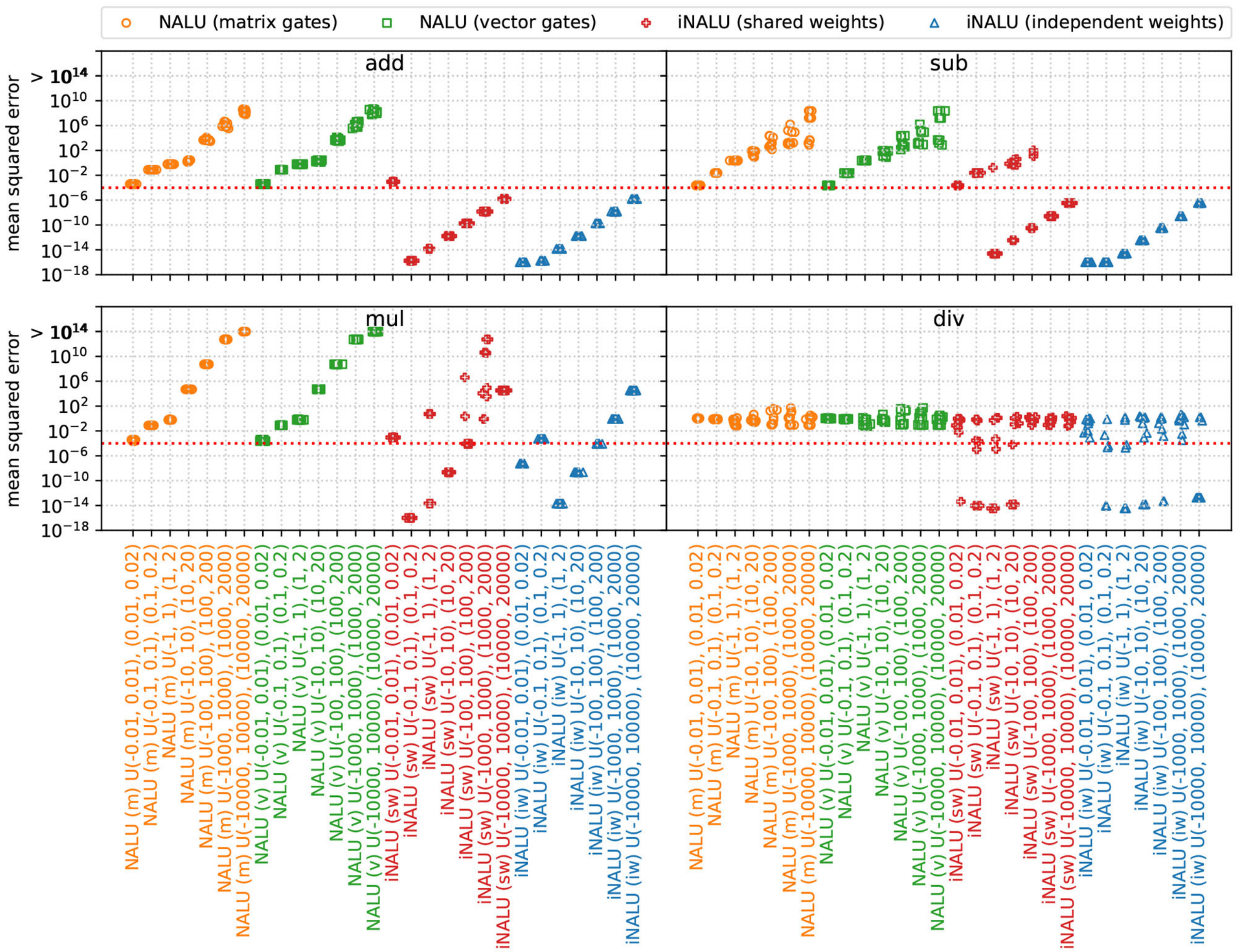

FIGURE 4 | MSE on extrapolation in the minimal arithmetic task for various uniformly distributed input magnitudes per operation. For a detailed description see **Figure 3**.

This means during training reinitialization can occur up to nine times. Note that this method could lead to incompletely trained models if a reinitialization occurs late during training in favor of a fair model comparison.

4.3. Experiment 1 - Minimal Arithmetic Task

Experiment 1 constructs the most minimalistic task where the model has two inputs and one output and analyzes the influence of the input value distribution by sampling *a* and *b* from uniform, truncated normal and exponentially distributed random variables in various ranges.

4.3.1. Results

The extrapolation results of this experiment are presented in **Figure 3**.

In general our iNALU models perform substantially better on all operations. With the exception of exponentially distributed data for $\lambda = 0.2$, for summation all and for subtraction almost all models succeed. For multiplication iNALU with independent weights performs best reaching very good precision with the exception of $E(0.2)$ and $N(-4, 2)$. All models yield worse results for division. In fact, for the original NALU, no tested input parameter configuration leads to acceptable MSEs (the average MSE is $4.36 \cdot 10^4$). Our models also yield mixed results, some solving the task nearly perfect after one to six reinitialization but others failing after nine reinitialization as well.

4.4. Experiment 2 - Input Magnitude

In this experiment we generate data of different magnitude for the minimal arithmetic task of experiment 1 to examine the influence of the data magnitude on the model precision. We sample *a* and *b* from a uniform random variable symmetrically around 0 from $(min, max) = (-10^{-2}, 10^{-2})$ to $(-10^4, 10^4)$. For each configuration we extrapolate to $(max, 2 \cdot max)$.

4.4.1. Results

The results of this experiment are shown in **Figure 4**. For input data of a magnitude larger than 1, the NALU models fail to capture the underlying function precisely for all operations. In contrast, the iNALU models calculate precisely for all magnitudes

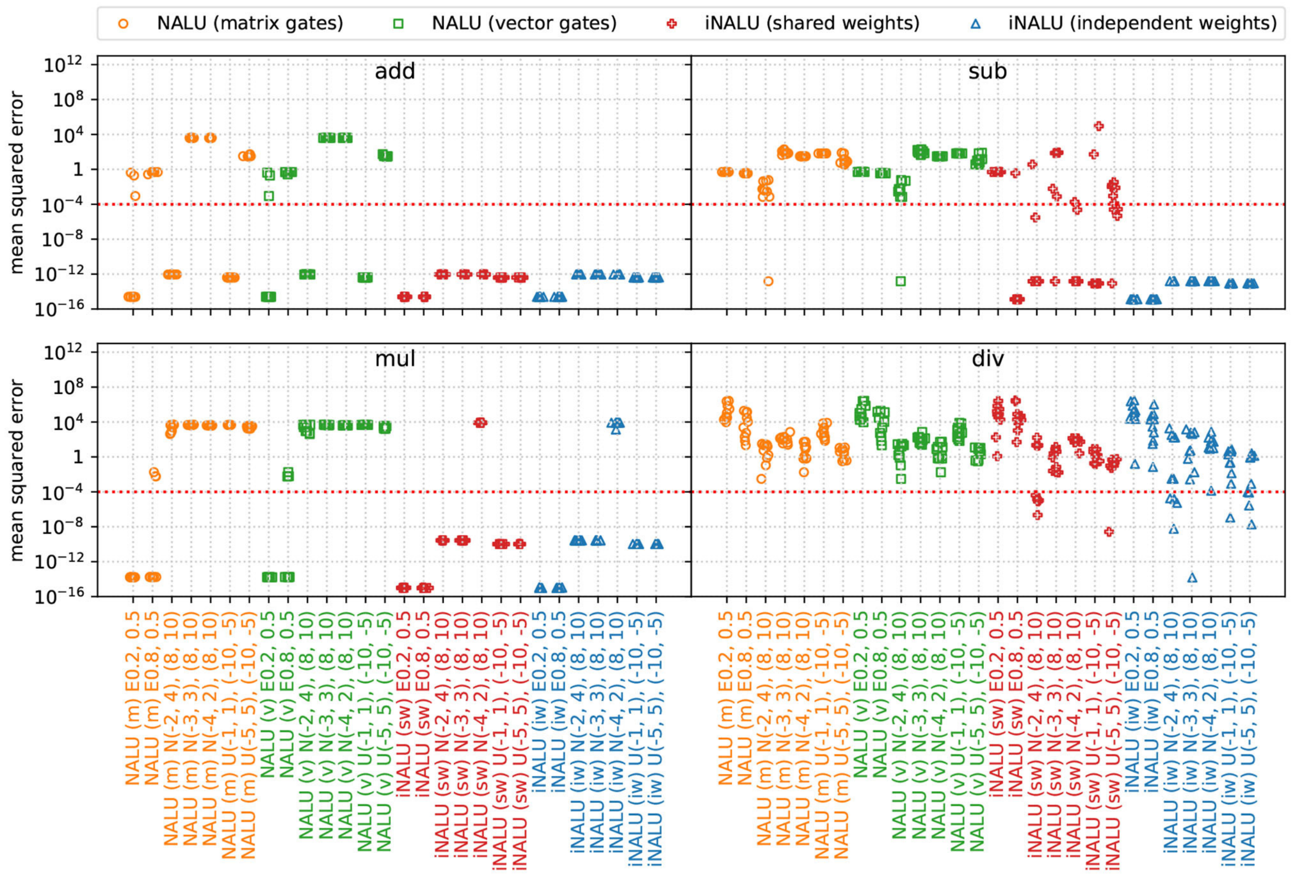

FIGURE 5 | MSE for various input distributions per operation over the extrapolation test dataset of experiment 3 (simple arithmetic task). For a detailed description see **Figure 3**.

for summation and subtraction. For multiplication the influence of the input data magnitude is larger, which was to be expected since the magnitude of the results for $a = 10^x, b = 10^y, a \times b$ is 10^{x+y} and so is the magnitude of the error, which is, as we report the mean square error, squared in addition. For division independently of the data magnitude, some iNALU models capture the underlying operation very precisely, others fail. All NALU models fail to calculate division precisely.

4.5. Experiment 3 - Simple Arithmetic Task

Experiment 3 is a generalization of the minimal arithmetic task where the model has to learn to ignore irrelevant input dimensions to calculate the correct solution.

This setting is motivated by real world tasks like spreadsheet calculations where one column is calculated by applying a simple operation to two specific columns while other columns are present but must not influence the result.

The model consists of one NALU layer with ten inputs and one output. We test the same input distributions as in the minimal arithmetic task (see section 4.3).

4.5.1. Results

Figure 5 shows the results of this experiment. Although, the setting of experiment 3 is slightly more complex than experiment 1, most performance patterns repeat. In the following, we want to highlight some interesting exceptions.

For input data sampled from an exponential distribution, the results improve for the original NALU models especially for summation and multiplication. For summation training is unstable, since some models succeed but others fail to learn the task. In contrast to the minimal arithmetic task, iNALU succeeds for summation of exponentially distributed data with $\lambda = 0.2$ and shows better results for multiplication. For division the situation of unstable training as discussed before even worsens such that only very few of our iNALU models succeed ($\approx$ 6.4% of all experiments reach a MSE $< 10^{-5}$). The original NALU failed constantly for division. For subtraction, our model with shared weights is slightly more unstable but our model with independent weights still yields stable results and calculates precisely.

4.6. Experiment 4 - Influence of Initialization

Experiment 1 suggests that training is unstable for some operations (subtraction and division). Whereas some of our improved models happen to solve the minimal task flawlessly, others fail to converge. As a consequence, suitable initialization seems to be crucial for successful training of more complex architectures. This fact is also confirmed by Madsen and Rosenberg Johansen (2019).

In this experiment, we analyze the effect of different parameters for random weight initialization of the neurons.

In contrast to the Minimal Arithmetic Task, the variables a and b are constructed by summing up 100 input vector entries assigned to a and b. Since Trask et al. (2018) doesn't specify the assignment in detail, we construct it by randomly assigning entries mutually exclusive to a and b and demand some inputs to be ignored by the model (since they neither contribute to a nor to b). We decide on the assignment once per task randomly such that the assignment is constant for all samples. Note, that the assignment is not an additional input to the neural network but instead it has to learn this assignment.

For this study, we examine the model performance of our iNALU model with shared weights for standard normal distributed input values such that $\mathcal{P} = \mathcal{N}$ and $\lambda = (\mu, \sigma) = (0, 1)$. We choose to initialize the model weights following a normal distribution as well. To find suitable initialization parameters, we performed an exhaustive search for the parameters $\mu_g, \mu_{\hat{M}}, \mu_{\hat{W}} \in \{-1, 0, 1\}$ and $\sigma_g, \sigma_{\hat{M}}, \sigma_{\hat{W}} \in \{0.1, 0.5\}$. We repeat each parameter setting 20 times with different seeds to be able to asses the model stability. Note that initializations which are too large bias the model toward specific operations, but especially sigmoid activations suffer from small random initializations (Glorot and Bengio, 2010).

4.6.1. Results

Table 1 shows the results of our parameter search. We consolidated the results for $\sigma = 0.1$ and $\sigma = 0.5$, since both parameters yielded similar results and report the maximum MSE of all runs for each parameter setting. This is a very strict evaluation metric since only 1 of 20 models failing could obfuscate 19 successful runs. However, we are particularly interested in parameters which lead to stable models. The results support our finding from the arithmetic experiments that division is very unstable to learn. To be precise, no model solved the problem for all parameter configurations and repetitions. Stable parameter configurations could be found for the remaining operations. Overall the configuration $(\mu_g, \mu_{\hat{M}}, \mu_{\hat{W}}) = (0, -1, 1)$ is clearly most stable among all tested parameters for this task and architecture.

4.7. Experiment 5 - Simple Function Learning Task

For the Simple Function Learning Task, we keep the setting of the previous experiment but focus on the comparison of our model using both, combined path-weights and separated path-weights to the originally proposed NALU in both variants (see section 3.1).

Since we found suitable initializations, we sample from uniform and truncated normal distribution and interpolate within the interval $[a, b] = [-3, 3]$ for both. This translates to a standard normal distribution ($\mu = 0, \sigma = 1$) for the truncated normal distribution. For the extrapolation interval we choose $[3, 4]$ and $[-5, -3]$ to test positive as well as negative values outside the training range with different standard deviations.

TABLE 1 | Maximum MSE over all models for the Simple Function Learning Task (extrapolation) for weight initializations means of −1, 0, 1.

E[G]	*E*[$\hat{M}$]	*E*[$\hat{W}$]	ADD	DIV	MUL	SUB
−1	−1	−1	1E−01 (93)	7E+09 (0)	1E+07 (81)	1E−02 (95)
		0	1E−02 (95)	7E+09 (0)	1E+07 (95)	1E−03 (98)
		1	3E+00 (98)	7E+09 (0)	**1E−04** (100)	**2E−08** (100)
	0	−1	3E+07 (13)	2E+14 (0)	1E+07 (25)	1E+04 (16)
		0	1E−01 (78)	7E+09 (0)	1E+07 (95)	1E−01 (68)
		1	5E+03 (73)	1E+05 (0)	**1E−04** (100)	3E−02 (89)
	1	−1	6E+07 (0)	5E+14 (0)	1E+07 (50)	8E+03 (0)
		0	9E+14 (30)	3E+06 (0)	1E+07 (87)	9E+14 (21)
		1	1E+17 (13)	7E+09 (0)	6E+00 (94)	1E+15 (14)
0	−1	−1	2E−01 (91)	7E+09 (0)	1E+07 (53)	1E−02 (95)
		0	1E−01 (88)	1E+05 (0)	1E+07 (64)	1E−02 (94)
		1	**1E−04** (100)	4E+05 (0)	**1E−04** (100)	**1E−04** (100)
	0	−1	8E+03 (6)	3E+14 (0)	1E+07 (29)	8E+03 (7)
		0	3E−01 (68)	1E+14 (0)	1E+07 (65)	2E−01 (65)
		1	2E−01 (71)	7E+09 (0)	**2E−04** (100)	3E+00 (70)
	1	−1	8E+03 (6)	7E+14 (0)	1E+07 (27)	7E+03 (0)
		0	3E+16 (23)	2E+14 (0)	1E+07 (60)	1E+15 (10)
		1	2E+17 (21)	7E+09 (0)	1E+01 (94)	4E+15 (18)
1	−1	−1	1E−02 (92)	4E+05 (0)	1E+07 (40)	1E−02 (98)
		0	9E−03 (93)	7E+09 (0)	1E+07 (50)	5E−03 (87)
		1	**2E−04** (100)	7E+09 (0)	**1E−04** (100)	6E−03 (97)
	0	−1	8E+03 (21)	2E+14 (0)	1E+07 (29)	8E+03 (34)
		0	3E−01 (36)	7E+09 (0)	1E+07 (36)	5E−01 (26)
		1	3E+00 (80)	7E+09 (0)	**2E−04** (100)	1E−01 (72)
	1	−1	4E+05 (11)	4E+14 (0)	1E+07 (61)	8E+03 (10)
		0	7E+16 (17)	7E+09 (0)	1E+07 (28)	1E+13 (0)
		1	2E+17 (21)	2E+14 (0)	1E+01 (93)	7E+15 (21)

Successful configurations (maximum loss < 0.001) in bold, percentage of successful repetitions in brackets.

4.7.1. Results

Figure 6 shows, that our iNALU models outperforms the original NALU for summation, subtraction and multiplication on almost all runs. Our model with independent weights is most promising since almost all runs succeed. However, few outliers indicate that the stability problem is not completely solved yet. This especially holds for division where all models fail to learn the operation correctly.

5. DISCUSSION

The experiments in section 4 analyzed the ability of the original NALU and our iNALU to solve various mathematical tasks and show that the performance of the NALU heavily depends on the distribution of the input data. The quality of the iNALU also depends on the input distribution but is in general more stable and achieves better results. For larger magnitudes of input data, multiplication becomes challenging for the iNALU however,

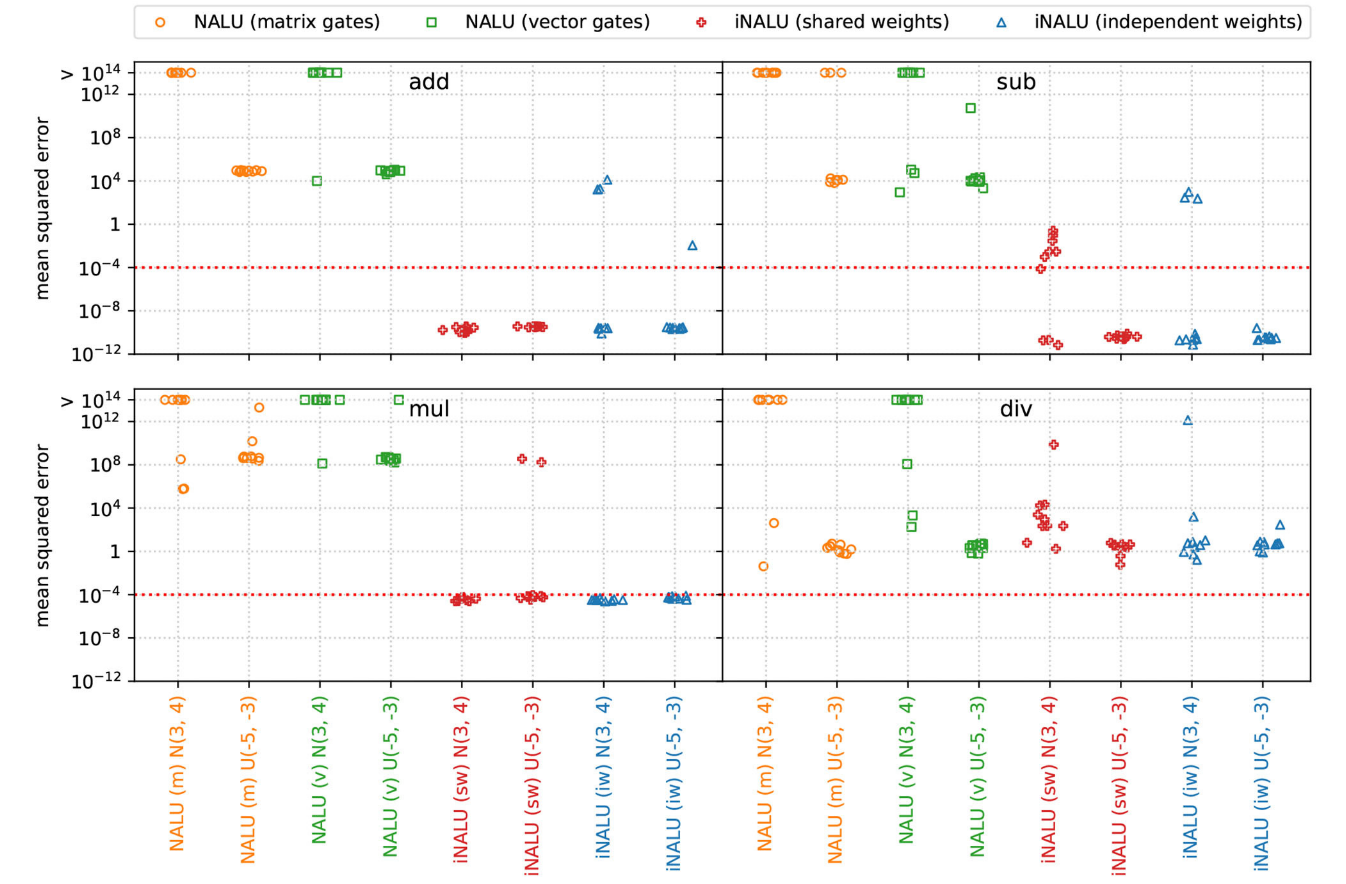

FIGURE 6 | Extrapolation MSE for Experiment 5 (simple function learning task). Original NALU with gating matrix (m) and gating vector (v) are colored orange and green, our iNALU model with shared weights (sw) is colored red and with independent weights (iw) in blue.

compared to the NALU the input range for which the model can multiply precisely is several magnitudes larger. Experiment 3 extends the arithmetic task by switching off several inputs. The results reinforce the findings of the first experiment that iNALU achieves better and more stable results than NALU. The differences between both iNALU models can be explained by the separate weighting matrix for summation/subtraction and multiplication/division. In experiment 5, the iNALU achieves for three of four operations acceptable results whereas the original NALU fails for all four operations.

In general, the MSE calculated on the extrapolation datasets provides a good intuition if the NALU has learned the correct logical structure which is resilient to other value ranges. The interpolation results are very similar regarding the relative performance of all models but in general achieve a higher precision and thus a lower MSE (e.g., for summation in experiment 1 our iNALU model with independent yields $6.14 \cdot 10^{-15}$ for interpolation and $5.45 \cdot 10^{-13}$ for extrapolation on average MSE).

Further, all experiments show that the operation division is the most challenging task for NALU and iNALU. The instabilities for division might be explained by the special case of dividing by near-zero and the sampling strategy for a and b: For sampling inputs in an interval including 0, division might cause huge or very small results depending on the assignments of dividend or divisor which are represented by completely different weights. Possibly irrelevant input variables might therefore influence the result by such magnitude that there is no clear gradient signal for the assignment.

Another observation is that the optimal initialization is dependent on many factors such as task, model size and value range. We want to emphasize that our parameter study is not intended to raise a claim for generally finding the optimal parameters, but rather to find initialization parameters for this specific task to allow a model comparison. Our study suggests the parameter configuration $(\mu_g, \mu_{\hat{M}}, \mu_{\hat{W}}) = (0, -1, 1)$ which seems to be reasonable, since it treats the summative/subtraction path and multiplicative/division path equally at beginning and assigns small activation weights to all inputs. We believe that the problem of generally finding optimal or near optimal initializations is an interesting and theoretically challenging task for future work.

6. CONCLUSION

Recently, the NALU architecture was proposed to learn mathematical relationships, which are necessary for solving various machine learning tasks. In this paper, we proposed an improved version of this architecture called iNALU. The original NALU is only able to calculate non-negative results for multiplication and division by design and often fails to converge to the desired weights. We solved the issues of multiplying and dividing with mixed-signed results and proposed architectural variants for shared and independent weights with input independent gating. Further, we introduced a regularization term and a new reinitialization strategy which help to overcome the problem of unstable training.

We evaluated the improvements in four large scale experiments which examine the influence of different input distributions and task-unrelated inputs. The first two experiments analyze the basic capabilities of NALU and iNALU. Further, the parameter study for the Simple Function Learning Task shows that the choice of weight initializations has a huge impact on model stability. The parameter study revealed suitable initialization parameters. We showed that our proposed architectures can learn simple mathematical functions and outperforms the reference models in terms of precision and stability.

Future work encompasses analyzing the stability issue from a theoretical point of view and evaluating the extensions in various downstream tasks. Last but not least, we want to improve the division in more complex learning scenarios.

DATA AVAILABILITY STATEMENT

The datasets presented in this study can be found in online repositories. The names of the repository/repositories and accession number(s) can be found below: https://github.com/daschloer/inalu/.

AUTHOR CONTRIBUTIONS

DS, MR, and AH contributed conception and design of the study. DS carried out the experiment supported by MR. DS wrote the first draft of the manuscript. All authors contributed to manuscript revision, read and approved the submitted version.

FUNDING

This work was partly funded by the Federal Ministry of Education and Research of Germany as part of the DeepScan project (01IS18045A) and the Bavarian Ministry of Economic Affairs Regional Development and Energy through the OBELISK project (IUK624/002). MR was supported by the BayWISS Consortium Digitization. This publication was supported by the Open Access Publication Fund of the University of Wuerzburg.

REFERENCES

Arjovsky, M., Chintala, S., and Bottou, L. (2017). "Wasserstein generative adversarial networks," in *Proceedings of the 34th International Conference on Machine Learning* (Sydney), 214–223.

Bolton, R. J., and Hand, D. J. (2002). Statistical fraud detection: a review. *Stat. Sci.* 17, 235–249. doi: 10.1214/ss/1042727940

Chen, K., Dong, Y., Qiu, X., and Chen, Z. (2018). Neural arithmetic expression calculator. *arXiv [Preprint]. arXiv:1809.08590.*

Freivalds, K., and Liepins, R. (2018). "Improving the neural GPU architecture for algorithm learning," in *Workshop on Neural Abstract Machines & Program Induction (NAMPI)* (Stockholm: ICML).

Garcia, S., Grill, M., Stiborek, J., and Zunino, A. (2014). An empirical comparison of botnet detection methods. *Comput. Security* 45, 100–123. doi: 10.1016/j.cose.2014.05.011

Glorot, X., and Bengio, Y. (2010). "Understanding the difficulty of training deep feedforward neural networks," in *International Conference on Artificial Intelligence and Statistics* (Sardinia), 249–256.

Kaiser, L., and Sutskever, I. (2016). "Neural GPUs learn algorithms," in *International Conference on Learning Representations* (San Juan).

Kalchbrenner, N., Danihelka, I., and Graves, A. (2015). Grid long short-term memory. *arXiv [Preprint]. arXiv:1507.01526.*

Kingma, D. P., and Ba, J. (2015). "ADAM: a method for stochastic optimization," in *International Conference on Learning Representations (ICLR)* (San Diego).

Lopez-Rojas, E., Elmir, A., and Axelsson, S. (2016). "PaySim: a financial mobile money simulator for fraud detection," in *European Modeling and Simulation Symposium (EMSS)* (Larnaca: Dime University of Genoa), 249–255.

Madsen, A., and Rosenberg Johansen, A. (2019). "Measuring arithmetic extrapolation performance," in *33rd Conference on Neural Information Processing Systems, NeurIPS 2019* (Vancouver, CA).

Reed, S., and De Freitas, N. (2015). Neural programmer-interpreters. *arXiv [Preprint]. arXiv:1511.06279.*

Ring, M., Schlor, D., Landes, D., and Hotho, A. (2019). Flow-based network traffic generation using generative adversarial networks. *Comput. Security* 82, 156–172. doi: 10.1016/j.cose.2018.12.012

Trask, A., Hill, F., Reed, S. E., Rae, J., Dyer, C., and Blunsom, P. (2018). "Neural arithmetic logic units," in *Advances in Neural Information Processing Systems 31*, eds S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (Curran Associates, Inc.), 8035–8044. Available online at: http://papers.nips.cc/paper/8027-neural-arithmetic-logic-units.pdf

Xie, W., Noble, J. A., and Zisserman, A. (2018). Microscopy cell counting with fully convolutional regression networks. *J. Comput. Methods Biomech. Biomed. Eng. Imaging Visual.* 6, 283–292. doi: 10.1080/21681163.2016.1149104

Zaremba, W., and Sutskever, I. (2014). Learning to execute. *arXiv [Preprint]. arXiv:1410.4615.*

Zhang, C., Li, H., Wang, X., and Yang, X. (2015). "Cross-scene crowd counting via deep convolutional neural networks," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (Boston), 833–841. doi: 10.1109/CVPR.2015.7298684

Conflict of Interest: The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.