INVESTIGATION OF THE NEURAL ARITHMETIC LOGIC UNIT

Kasper P. Rolsted, Frederik R. Warburg, Harald L. Mortensen

{s152987, s153847, s154436}@student.dtu.dk

ABSTRACT

Neural networks have proven advantageous in a wide variety of fields; including object detection, speech recognition, language processing. With a hunger for data, these networks can learn complex functions and interpolate really well. However, these networks tend to have problems when presented with data outside the training domain, often resulting in poor generalization capabilities. Recently, the Neural Arithemtic Logic Unit (NALU)[1] has shown promising capabilities of extrapolating well beyond the training domain on numerous experiments. This paper seeks to investigate the NALU by replicating experiments from the original paper while exploring the potential problems this unit in prone to experience. Experiments show that the NALUs generally are hard to train, even when given simple problems such as addition of two numbers. In a variety of settings, it is seen how the presented gating function is likely to be the cause of poor extrapolation, especially given negative numbers. In some cases, however, the NALU shows promising results, outperforming traditional MLP's for tasks on positive numbers. Several experiments are conducted in order to train the model more smoothly, none of which showed great increase in stability. Implementation and experiments can be found at `https://github.com/FrederikWarburg/latent_disagreement`.

1. INTRODUCTION

When children learn a language, they do not just memorize certain phrases. They learn the underlying concepts and relations between words, sentences and context. By learning this underlying logic about a language, humans are able to learn from relatively few examples and extrapolate to form sentences, by combining words they have not previously heard together. As such, humans learn a systematic compositionality[2] that makes them able to generalize beyond the scope of a limited number of examples. This human ability, to learn from few examples, is not limited to language, but extends to all areas of human intelligence.

Recent improvements in Artificial Intelligence (AI) has risen from deep learning, where increasing amounts of data and computational capabilities has enabled Deep Neural Networks to learn very complex functions. These models have shown incredible capabilities of memorizing complicated functions in fields such as Speech Recognition [3] [4], Object Detection [5] [6] [7] and Natural Language Processing [8]. As opposed

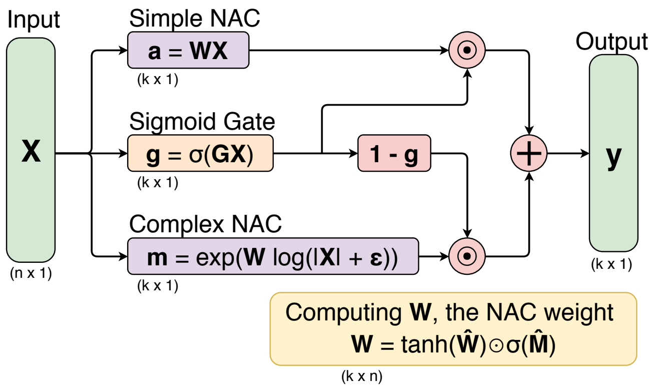

Fig. 1: The figure shows an alternative visualization of the NALU to the one provided in [1]. The input is forwarded through the complex and simple NAC, as well as a gate $\mathbf{g}$. The gate decides the weighting of the simple and complex NAC. The learnable parameters are $\hat{\mathbf{W}}$, $\hat{\mathbf{M}}$ and $\mathbf{G}$.

to humans, state-of-the-art Deep Neural Networks require huge amounts of data and show little understanding of the actual underlying concepts - suggesting a lack of systematic compositionality, algebraic understanding and "logic mind" [9]. As an example, Neural Networks has shown to perform poorly when exposed to systematic differences between the training and test data [1]. Thus, the predictive capabilities of these models are confined to the training range, and extrapolation outside the training range is in general a very challenging task, resulting in bad generalization.

Inspired by the Arithmetic Logic Units (ALU) found in computers, recent work [1] suggests a Neural ALU (NALU) that has shown great promise in learning numeric reasoning. Trask et al. showed that inserting the NALU in a neural network improved its capabilities of extrapolating data outside the training region. Our paper will largely build on top of this work. Our main contributions consist of the following:

- We provide an in-depth review of the mathematical formulation of the NALU.
- We conduct an extensive experimental analysis on the interpolation and extrapolation capabilites of the NALU.
- We formulate three potential alterations of the NALU architecture and investigate the interpolation and extrapolation capabilities of these.

Technical University of Denmark, DTU Compute

The paper is structured as follows: In Section 2, related works and current approaches for numeric reasoning are described. Following in Section 3, we provide an explanation of the intuition and the mathematical foundation of NALU. In Section 4 the experimental setup for learning simple functions and subset selection is described. In Section 5, the results from these experiments are presented. We end with a discussion in Section 6 and a conclusion in Section 7. For implementation and experiments details consult our GitHub repository `https://github.com/FrederikWarburg/latent_disagreement`.

2. RELATED WORKS

In a review of the recent progress in AI and cognitive science, Lake et al.[10] highlights that current deep neural networks learn and think very differently than humans. Lake et al. point towards humans' ability to generalize from a very few examples and learn concepts rather than patterns. Similarly, Lake et al. in [2] explored the generalizational capabilities of several different Recurrent Neural Network (RNN) architectures, utilizing sequence-to-sequence methods. Their general discovery was the networks inability to extract systematic rules from the training data, putting emphasis on the problem that training and test data has to come from similar distributions. As such, this lack of systematicity could be responsible for neural networks hunger for data, in order to generalize. Similar work was done in [11], also emphasizing the generalization issues of state-of-the-art neural networks.

In other works, an attempt was made to explicitly imitate human and animal learning. Bengio et al. suggested Curriculum Learning [12] where the basic idea is to begin training with small and easier sub-tasks, and then gradually increase the difficulty level, instead of randomly presenting tasks, thus replicating the learning stages of humans to some extent. They found that curriculum learning could both increase training speed and improve generalization, suggesting that the model learns the underlying structure of the data much better. In [13] Zaremba et al. investigate the capabilities of a RNN with Long Short-Term Memory (LSTM) units to learn logical operations by evaluating short computer programs. An LSTM was trained to read a short program character-by-character and output the results of the program. Zaremba et al. found that it was necessary to impose curriculum learning for the model to learn the logic behind nested statements.

As an alternative to focusing on the learning processes, recent research has focused on imposing different units in networks for better generalization and ability to learn simple underlying concepts. Building on top of the Equation Learner (EQL) presented in [14], Sahoo et al. presents the model EQL$^{\div}$ that aims at discovering simple underlying relationship between input and output data [15]. Through experimental work, Sahoo et al. found that for systems governed by analytical expressions, their improved Equation Learner (EQL$^{\div}$) Unit is able to identify the underlying equation and extrapolate to unseen domains. Sahoo et al. shows that their EQL$^{\div}$ is able to learn formulas with division, which had previously yielded difficulties.

Also in this category of approaches, Trask et al. presented the Neural Arithmetic Logic Unit (NALU) [1] that imitates the Arithmetic Logic Unit (ALU) of computers, to handle logical statements. Similar to Sahoo et al., they showed that the NALU was able to learn simple functions and extrapolate outside the training region. Through experimental work, Trask et al. showed that the NALU can be incorporated in both RNNs and CNNs. Utilizing these models in a variety of experiments, they found that the networks' ability to extrapolate significantly exceeded that of conventional architectures - showing that it is in fact possible for neural nets to learn the underlying concept and move beyond the training space for more complex functions.

3. METHODS

In this section we describe the Neural Accumulator (NAC) and Neural Arithmetic Logic Units (NALU) presented in [1] and we seek to give an overview of these reasoning behind the proposed unit.

The central component of the NALU is the NAC unit, an accumulator, which is designed to be able to add and subtract numbers. It does this using simple matrix multiplication on the input data, thus the simple NAC is given as

$$\mathbf{a} = \mathbf{W}\mathbf{x} \tag{1}$$

where $\mathbf{W}$ is a matrix of size $K \times N$, $\mathbf{x}$ is the input with size $N \times 1$ and thus $\mathbf{a}$ becomes $K \times 1$. In general $\mathbf{x}$ could be any size, but we shall consider $N \times 1$ for clarity and increased intuition.

The clever part in the NAC, is how the weighting matrix $\mathbf{W}$ is defined. In order to perform simple unweighted addition, subtraction and be able to ignore values in the input, the weight matrix $\mathbf{W}$ must be constrained to the values $\{-1, 0, 1\}$. To accomplish this, the weighting matrix is defined as

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

where $\hat{\mathbf{W}}$, $\hat{\mathbf{M}}$ are matrices of size $K \times N$. The activation functions $\tanh$ and σ have saturation points in respectively $\{-1,\ 1\}$ and $\{0, 1\}$. Thus, each element in the learned parameter $\mathbf{W}$ will be close to their combined saturation points $\{-1,\ 0,\ 1\}$. By having exactly these saturation points, the NAC can learn to imitate an operator which can add, subtract or ignore elements, thus imitating discrete choices. A large benefit of constructing $\mathbf{W}$ in this fashion, is the fact that it is differentiable, which it would not have been if it had been simply constrained to the discrete choice of values $\{-1, 0, 1\}$. Since the derivatives of $\mathbf{W}$ can be computed, the weights can be optimized using backpropagation.

The complex NAC unit, $\mathbf{m}$, is the simple NAC transformed into log-space. Thus, it is simply the exponential of the simple NAC where the input is the logarithm of the input. This means that the complex NAC turns addition and subtraction

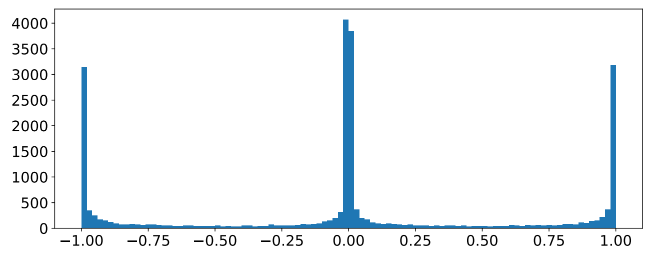

Fig. 2: The distribution of $\tanh(\hat{\mathbf{W}}) \odot \sigma(\hat{\mathbf{M}})$ shows the rather distinct saturation points at $\{-1, 0, 1\}$ These distinct saturation points are favorable as we wish to estimate discrete properties.

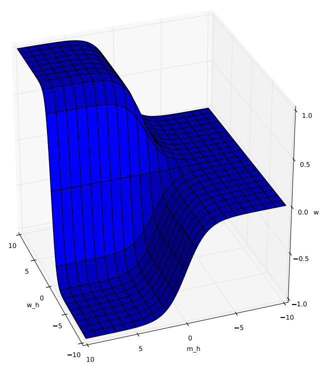

Fig. 3: The optimization surface $\mathbf{W}$ plotted as a function of $\hat{\mathbf{W}}$ and $\hat{\mathbf{M}}$. The saturation points $\{-1, 0, 1\}$ are the results of the three planar surfaces.

into multiplication and division, given the nature of the transformation to log-space. The complex NAC is given as

$$\mathbf{m} = \exp \mathbf{W}(\log(|\mathbf{x}| + \epsilon)) \tag{3}$$

where ϵ is included to increase numerical stability by avoiding $\log(0)$.

To illustrate an example of how division is carried out by transforming the data to the log-domain, let us consider the following simple instance: $\mathbf{x} = [x_1 \quad x_2]^T$, $\mathbf{W} = [1 - 1]$. The simple NAC would compute $\mathbf{a} = \mathbf{W}\mathbf{x} = x_1 - x_2$. In the complex case, the NAC would compute $\mathbf{m} = \exp(\log(|\mathbf{a}| + \epsilon)) = \exp(\log(|x_1| + \epsilon) - \log(|x_2| + \epsilon)) = \exp\left(\log\left(\frac{|x_1|+\epsilon}{|x_2|+\epsilon}\right)\right) = \frac{|x_1|+\epsilon}{|x_2|+\epsilon}$, utilizing the simple domain changes of exp and log.

This clearly illustrates that the complex NAC in itself is unable to learn division (and multiplication) when the signs of x_1 and x_2 are opposite, due to the absolute-value term. The transformation to the log-exp domain introduces some numeric instability that lead to Trask et al.'s poor performance on division tasks[1]. We perform an analysis on this instability in Appendix B

To be able to decide whether to perform addition / subtraction or multiplication / division, a weighting matrix $\mathbf{G}$ is learned as a part of the gating function $\mathbf{g} = \sigma(\mathbf{G}\mathbf{x})$. Due to the fact that the sigmoid function has saturation points at 0 and 1, as the network is trained, $\mathbf{G}$ should approach values such that each index of $\mathbf{g}$ should go towards either 0 or 1. This should make the network "choose" between either the simple NAC $(+/-)$ or the complex NAC $(*/\%)$. Thus the learned weightings are then

$$\mathbf{g} = \sigma(\mathbf{G}\mathbf{x}) \tag{4}$$

There is however no guarantee that the values of $\mathbf{g}$ actually become binarized, since there might be many local minima. This could cause $\mathbf{g}$ to not actually be "choosing" between the Simple NAC and the Complex NAC, but instead find a weighting bwetween the output of both. We illustrate examples of this in Appendix D.3.

To use the "gate" or weighting of $\mathbf{g}$, the output of NALU is given as the following weighted sum

$$\mathbf{y} = \mathbf{g} \odot \mathbf{a} + (1 - \mathbf{g}) \odot \mathbf{m} \tag{5}$$

As such, the NALU can learn functions that consist of addition, subtraction, multiplication and division. Since the combined values of $\mathbf{W}$ and $\mathbf{g}$ should capture simply the operation to perform, and be independent of the sign (at least in the case of the simple NAC) and magnitude of the input, the NALU should be able to utilize the learned parameters on data well beyond the training range [1].

4. EXPERIMENTAL SETUP

We have conducted several experiments to investigate the expressive capability of the NALU. We have used the experiment *"Simple Function Learning Task"*[1] as inspiration, where the input is a vector $\mathbf{x} \in \mathbb{R}^{100}$ from which we wish to select two subsets $a = \sum_{i=p}^{q} x_i$ and $b = \sum_{j=r}^{s} x_j$ where $0 \leq p < q < r < s < 100$[1]. This is followed by an operation, $a \circ b$, where $\circ$ is a simple arithmetic function such as addition, subtraction, multiplication or division. The left illustration in Figure 4 shows the original experimental setup, **Full Task**.

We found that this task was challenging to learn for the NALU. Therefore, we also experimented with its two-sub tasks; respectively selection of the subsets a and b (the middle illustration in Figure 4), and learning an operation on $a \circ b$ (the rightmost illustration), which we will refer to as respectively **Subset Selection Task** and **Arithmetic Task**. We test these sub-tasks to find which one is the main culprit in making the task hard to learn. The architecture for the NAC, NALU and MLP for each experiment is found in Appendix C.

[1]In theory, the subsets should be able to overlap, however, in a set of preliminary experiments, we found that the overlapping sets increased the difficulty of the task. Therefore, in this report, we have solely focus on the task where the subsets are not overlapping.

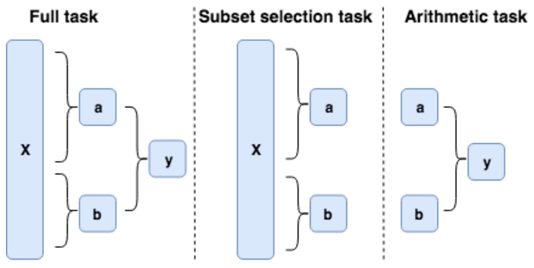

Fig. 4: Illustration of the three distinct experiments. Subset selection and operation **(left)**, only subset selection **(middle)** and only operation **(right)**.

To ease the understanding of the progress of the results (Section 5), we will briefly summarize the experiments and their conclusions. The goal of these experiments is to describe when and why the NALU succeeds or fails to either interpolate or extrapolate.

- **Experiment 1**: Interpolation and extrapolation for the NALU, NAC and MLP for the Full Task. Neither the MLP nor the NALU could extrapolate. The NAC extrapolates reasonably for addition and subtraction. Thus, we separately test the two sub-tasks; the Arithmetic Task and the Subset Selection Task.
- **Experiment 2**: Investigation of the performance of the NALU and NAC on the Arithmetic Task for different training ranges. Altering the training range did not improve the models extrapolation capabilities.
- **Experiment 3**: Experiment with the Subset Selction Task, which we expect to be more difficult than Arithmetic Task. Investigation of weight initialization and extrapolation capabilities for converged models. We see the NALU only converges on rare occasions (irregardless of initialization) and converged NALUs can only extrapolate to the positive range.
- **Experiment 4**: Three alterations of the gating function that potentially could improve the robustness for the Subset Selection Task are investigated: temperature, learnable bias parameters and input-independent gate. Neither of the suggested improvements seem to increase the robustness of the model.

5. RESULTS

In this section, we report the results from the experiments briefly described in Section 4. For each experiment, we intend to describe when and why the model failed or succeeded.

5.1. Experiment 1: Learning The Full Task

We train respectively two stacked NACs and NALUs as well as an MLP for the Full Task. For interpolation, we sample **x** from the Uniform distribution $U(0, 1)$. We see in Table 1 that the NAC can select and perform addition and subtraction, however fails for multiplication and division. This is as expected as the NAC does not operate in the log-space. On the other hand, the NALU should been able to learn multiplication and division, however, from the table, we see that it also fails on this task.

	Interpolation			**Extrapolation**		
Operation	NAC	NALU	MLP	NAC	NALU	MLP
$+$	**0.00**	0.42	0.25	**0.36**	inf	inf
$-$	**0.00**	0.11	0.19	**240.35**	inf	inf
$*$	4193.47	32.71	4.39	inf	inf	inf
$/$	inf	8558.42	14.39	inf	inf	inf

Tab. 1: Interpolation and extrapolation: MSE medianed over 10 experiments. We report inf, if the MSE exceeded 10^5. Interpolation is performed on U(0,1) and extrapolation on U(0,100).

In order to investigate if the models have learned the underlying structure of the data, we extrapolated by sampling the elements of **x** form an uniform distribution $U(0, 100)$. However, from Table 1 it is seen that only the NAC for addition has managed to learn this underlying structure. In an effort to improve these initial results, we adjust several of the model's hyper parameters (Appendix D.1). As neither of our hyper parameter configurations investigated in Appendix D.1 improved the performance of the NALU on the full task, we decided to investigate the two subtasks; subset selection task and arithmetic task respectively.

5.2. Experiment 2: Learning The Arithmetic Task

We investigate if the NALU can learn to add two numbers. In this experiment we trained the NALU model in variety of different ranges to test the training domain's impact on the extrapolation capability (Table 2). From this table we see that the NALU generally does not extrapolate well regardless of the training domain. It seems that training in a U(-1,1) is slightly better than U(0,1), however, these models' error are within the same order of magnitude. For further explanation of the experiment and the results see Appendix D.2.

	Training Range			
Extrapolation Range	$U(0, k = 1)$	$U(-1, k = 1)$	$U(0, k = 10)$	$U(-10, k = 10)$
$U(0, 5k)$	15.39	**9.96**	$2.2 \cdot 10^2$	66.91
$U(0, 10k)$	82.74	**48.86**	$2.5 \cdot 10^3$	$9.7 \cdot 10^2$
$U(0, 100k)$	$\mathbf{1.1 \cdot 10^3}$	$8.4 \cdot 10^3$	$2.9 \cdot 10^6$	$1.0 \cdot 10^6$
$U(-5k, 5k)$	39.30	**8.82**	$8.5 \cdot 10^2$	83.70
$U(-10k, 10k)$	$1.3 \cdot 10^2$	**39.44**	$4.0 \cdot 10^3$	$7.1 \cdot 10^3$
$U(-100k, 100k)$	$1.1 \cdot 10^4$	$\mathbf{7.0 \cdot 10^3}$	$2.4 \cdot 10^6$	$5.4 \cdot 10^5$

Tab. 2: MSE for extrapolation over 6 different ranges. Reporting best MSE over several trained models

Thus, based on Table 2 we conclude that the NALU have difficulties learning addition regardless on the training range. We found similar results for the other simple arithmetic operations. After studying the arithmetic function, we investigate how well the NALU performed on the other subtask - namely the subset selection task.

5.3. Experiment 3: Learning The Subset Selection Task

We want to investigate how well the NAC and NALU can select specific subsets of an input (Subset Selection Task described in Section 4) and possible causes of eventual failures.

5.3.1. *Initialization*

In a series of preliminary training attempts, we found that the ability for the NALU to learn subset selection is not stable. The NALU seemed especially vulnerable in regards to initialization of the weights, with no obvious system of when and how it would manage to converge. To investigate this phenomenon, we first train and test the NAC, NALU and MLP on the subset selection task with different initialization of the weights. Each experiment is repeated 10 times, and we count the number of experiments in which the validation loss has converged to below 10^{-6}.

Initialization	NAC	NALU	MLP
Kaiming Uniform	100 %	30 %	100 %
Xavier Normal	100 %	70 %	100 %
Kaiming Normal	100 %	50 %	100 %
Zeros	100 %	10 %	100 %
Ones	100 %	100 %	100 %

Tab. 3: Frequency of convergence

As illustrated in Table 3, we find that the NAC is able to converge and interpolate well for every initialization. The opposite case seems true for the NALU, as it does not converge and interpolate well for all initializations, with the notable exception of the Ones-initialization (discussed in Appendix D.4). The MLP interpolates very well as expected. By this experiment, it seems clear that in the extension from the NAC to the NALU, a great amount of instability is added. Since the $\mathbf{W}$-parameter is of the same size in both experiments, the blame is pointed at the inclusion of the Simple-/Complex-selector $\mathbf{g} = \sigma(\mathbf{Gx})$. This seems to be backed up when inspecting the movement of the $\mathbf{W}$ and $\mathbf{g}$ over their optimization surfaces, as illustrated in Appendix D.3.

Since there does not seem to be a significant difference between the other initializations, except it seems that zero-initialization is bad and ones-initialization is unfeasible (Appendix D.4), we want to continue to use the Kaiming Uniform initialization, as suggested in [1].

5.3.2. *Extrapolation*

For the models, which converges and interpolate well, we test how well these model generalizes to data outside the training domain. In this experiment, we consider the extrapolation capabilities of a converged NAC, NALU and MLP. Each model has managed to learn to interpolate well on $U(0, 1)$.

Extrapolation	NAC	NALU	MLP
$\boldsymbol{x} \in U(0, 1)$	**0**	**0**	$3.0 \cdot 10^{-4}$
$\boldsymbol{x} \in U(-0.001, 0.001)$	**0**	**0**	$5.8 \cdot 10^{3}$
$\boldsymbol{x} \in U(0, 100)$	**0**	**0**	$3.6 \cdot 10^{-2}$
$\boldsymbol{x} \in U(-100, 100)$	**0**	inf	$3.6 \cdot 10^{2}$
$\boldsymbol{x} \in U(-1000, 1000)$	**0**	inf	$3.6 \cdot 10^{4}$

Tab. 4: MSE: Extrapolation of NAC and NALU compared to a baseline MLP. MSE below 10^{-6} is set to 0 and MSE that exceeded 10^{6} is set to inf.

The great performance of the NAC is as expected, since $\mathbf{W}$ is not sensitive to neither the signs nor magnitude of $\mathbf{x}$. Since the NAC extrapolates well while the NALU does not, we are again led to believe that the cause of the underperformance of the NALU is the Simple-/Complex-selector $\mathbf{g}$. This is backed up by the grotesque magnitude of the losses, as this indicates that the selector has selected the Complex unit.

To alleviate this problem with $\mathbf{g}$, we wish to consider both changing the definition slightly by including a learnable bias parameter to increase its' generalization capabilities and to improve the actual learning process by including a temperature term, which should ensure that the values of $\mathbf{g}$ can move more freely in the training phase, until later where they will be "locked in". Consult Appendix A for an in-depth theoretical motivation for these changes.

5.4. Experiment 4: Improving The Gating Function

In this subsection, we describe and evaluate three potential improvements of the gating function. For a detailed theoretical argument on why these alterations could improve the model, consult Appendix A.1, Appendix A.2 and Appendix A.3.

5.4.1. *Temperature*

The results from the preliminary experiments showed considerable issues in learning the gating function $\mathbf{g}$, which evidently proves that training the NALU is difficult. For large magnitudes of $\mathbf{x}$, $\mathbf{g}$ risks being so large that it gets stuck in a flat plane. As we observe in Appendix D.3, it seems $\mathbf{g}$ sometimes has difficulty in dealing with the high-acceleration areas in the edge of the flat planes. As such, a temperature measure is included in the gating function, in order to smooth the sigmoid optimization surface, hopefully giving the gate the ability to learn more freely.

In order to smoothen the optimization space, the temperature term is included as a scalar, and we express the gating function as $\mathbf{g} = \sigma(\tau\mathbf{Gx})$. To ensure that the gate still resembles the original gate, we linearly increment the scalar across training, to eventually be equal to 1, yielding the original function. As such, the experiment conducted here, smoothes the optimization surface by initially choosing $\tau = 0.2$, and increasing τ by 0.0008 for each epoch. The results are reported as an average test loss over 10 unique experiment for each extrapolation range. All experiments were trained with $\mathbf{x} \in U(0, 1)$.

Extrapolation	**NALU**
$\mathbf{x} \in U(0, 1)$	0
$\mathbf{x} \in U(0, 10)$	0
$\mathbf{x} \in U(0, 100)$	0
$\mathbf{x} \in U(-10, 10)$	inf
$\mathbf{x} \in U(-100, 100)$	inf

Tab. 5: Extrapolation loss with temperature smoothing. Where a median MSE below 10^{-6} is rounded to 0 and above 10^6 is set to inf.

From Table 5, we see similar results compared to the original experiment (Table 4) conducted without temperature scaling. We still seem to be able to extrapolate within the positive range of x, but again the model struggle with the negative range. As such, including a temperature term does not increase the performance of our network significantly, and the problems with the negative range continue to exist.

5.4.2. *Bias*

We extend the model to have learnable bias parameters in the Simple-/Complex-selector $\mathbf{g} = \sigma(\mathbf{Gx})$, such that we instead define $\mathbf{g} = \sigma(\mathbf{Gx} + \boldsymbol{\beta})$. As described in Appendix A.2, the inclusion of bias parameters should allow for the model to actually learn proper subset selection. We train the NALU in different range and test its extrapolatory capabilities on wider ranges (Table 6).

	Training Range			
Extrapolation Range	$U(0, k=1)$	$U(-1, k=1)$	$U(0, k=10)$	$U(-10, k=10)$
$U(0, 5k)$	0	inf	inf	$6.87 \cdot 10^3$
$U(0, 10k)$	0	inf	inf	$1.37 \cdot 10^4$
$U(0, 100k)$	0	inf	inf	$1.39 \cdot 10^5$
$U(-5k, 5k)$	inf	inf	inf	$3.13 \cdot 10^2$
$U(-10k, 10k)$	inf	inf	inf	$5.92 \cdot 10^2$
$U(-100k, 100k)$	inf	inf	inf	$5.82 \cdot 10^3$

Tab. 6: Shows the extrapolatory capabilities of the NALU with learnable bias parameters in the gate. Table reports average MSE over 10 runs. If MSE less than 10^{-6} we set to 0, and if MSE exceeds 10^6 we set it to inf.

From Table 6 we see clearly that with the learnable bias parameters, only the NALU trained on $\mathbf{x} \in U(0, 1)$ manages to extrapolate, but only then to the positive range. None of the other manage to extrapolate in any meaningful fashion. As such, this addition to the model was of no benefit.

The intended outcome was for increased stability of the algorithm in regards to the signs and magnitude for $\mathbf{x}$, but we clearly do not see this. This must stem from the fact that the model with the bias parameter is not "incentivized" to move towards our perceived ideal solution of $\mathbf{G} = \mathbf{0}$, $\boldsymbol{\beta} >> \mathbf{0}$, but generally finds a local minimum solution instead. This could perhaps be remedied by some controlling factor or function which moves $\mathbf{G}$ towards 0 or increases the importance of the learnable bias parameter.

5.4.3. *Input independent gating function*

As a final experiment to combat the troubles with the learned gate, we propose an input independent gating function. The reason behind this is that as of now, the gate has been dependant on the input through $\mathbf{g} = \sigma(\mathbf{Gx})$ or $\mathbf{g} = \sigma(\mathbf{Gx} + \boldsymbol{\beta})$, which evidently fails too often. As such, similar work suggests defining an input independent gating function as $\mathbf{g} = \sigma(\boldsymbol{\beta})$. We conduct the same experiment as before, now with input independent gating function. The average MSE loss for 10 models for each extrapolation range are reported in Table 7.

	Training Range			
Extrapolation Range	$U(0, k=1)$	$U(-1, k=1)$	$U(0, k=10)$	$U(-10, k=10)$
$U(0, 5k)$	$7.34 \cdot 10^{23}$	$5.95 \cdot 10^{31}$	$3.39 \cdot 10^{8}$	$2.72 \cdot 10^{1}$
$U(0, 10k)$	$4.36 \cdot 10^{62}$	$4.39 \cdot 10^{68}$	$7.90 \cdot 10^{12}$	$4.40 \cdot 10^{6}$
$U(0, 100k)$	$6.91 \cdot 10^{170}$	$1.95 \cdot 10^{195}$	$2.16 \cdot 10^{16}$	$7.76 \cdot 10^{27}$
$U(-5k, 5k)$	$5.35 \cdot 10^{27}$	$2.50 \cdot 10^{31}$	$2.05 \cdot 10^{10}$	$2.52 \cdot 10^{1}$
$U(-10k, 10k)$	$2.88 \cdot 10^{59}$	$2.71 \cdot 10^{68}$	$4.25 \cdot 10^{10}$	$6.02 \cdot 10^{6}$
$U(-100k, 100k)$	$6.14 \cdot 10^{172}$	$4.45 \cdot 10^{199}$	$8.72 \cdot 10^{16}$	$6.88 \cdot 10^{27}$

Tab. 7: Average MSE loss over 10 models with input independent gating function.

The extrapolation capabilities of these models are very poor, despite that fact that the models converge on the training data in most of the experiments. It seems as if the model does not correctly capture the underlying structure when trained using this input-independent gate, but merely overfits on the given training data. As a result, the model incorrectly chooses the $true$ subsets, leading to severe extrapolation error.

6. DISCUSSION

In general, throughout this paper, we found that the NALUs were hard to train and often did not extrapolate well to data outside the training range. As such, the NALU's ability to interpolate and extrapolate does not seem to reach the level as described by Trask et al.[1]. To verify that this was not due to our implementation, we have compared and tested most of our experiments on several publicly available GitHub implementations of the NALU[2], [3], [4] that Trask himself has endorsed by forking. Thus, we are confident that the poor performance is not due to our implementation.

In our results, we found that the simple NAC was able to extrapolate to negative range, however, the NALU had difficulties learning the gating function $\mathbf{g}$. Therefore, we investigate several architectural alterations of the unit in order to learn the gating function more robustly.

Inspired by the Gumbel Softmax, we implemented an inclusion of a temperature term in the Simple-/Complex-selector $\mathbf{g}$. The intention of including a temperature term was to start out with a relaxed optimization surface for g, which is tightened as the model trains, thus keeping the model from being locked in until later in the learning process. We found that this did not seem to be the case and that there were no obvious benefits from including a temperature term.
We saw that the performance for extrapolation to positive ranges stayed much the same, with worse or equivalently

[2]`https://github.com/Nilabhra/NALU`
[3]`https://github.com/merrymercy/NALU`
[4]`https://github.com/kevinzakka/NALU-pytorch`

poor performance in the extrapolation to negative and positive range. This indicates that it is probably not the case that $\mathbf{g}$ gets locked in at some point, but simply that the learned weights of $\mathbf{G}$ are not capable of dealing with a change of signs, as discussed in Appendix A.2.

This lead to the extension of including learnable bias parameters in the Simple-/Complex-selector $\mathbf{g}$, intending to make it possible for the selector to actually be robust in regards to change of signs in $\mathbf{x}$. We found, however, that this did not actually improve on the extrapolation capabilites of the model and even seemed detrimental to learning. It seems that this instead lead to a higher degree of overfitting, as fewer of the models actually managed to converge. This seems to agree with what we discussed in Section A.2, as only the very specific solution of $\mathbf{G} = \mathbf{0}$ would lead to a proper generalizable model. Since there is nothing else in the model or learning process which would necessarily encourage this minimum over the many other local minima, we believe the addition of a bias did not do the model many favors.

From the interpretation of the results from the inclusion of a learnable bias parameter and the requirement that $\mathbf{G}$ should become $\mathbf{0}$, a natural extension is then simply to consider an input-independent $\mathbf{g}$, in order to avoid the many local minima. This attempt at improving the model failed completely, though, as the extrapolatory capabilites when trained on $U[0, 1]$ were nonexistent. By inspection of $\mathbf{G}$ for trained models, we in general found that it converged towards one large positive value and one large negative value, clearly indicating that the model no longer seems to learn the intended connection between input and output.

7. CONCLUSION & FUTURE WORK

We found that for strictly positive values of $\mathbf{x}$, the NALU is able to learn and extrapolate to a great extent in the combined task of subset selection and arithmetic operations. In stark contrast, however, we see that as soon as $\mathbf{x}$ can attain negative values, much of this strength breaks down. We found that learning the gating function was challenging. In attempt to remedy this, we attempted three different alterations to the NALU architecture, but found that none of them were adequate and some were even detrimental to the learning process and extrapolatory performance.

8. REFERENCES

[1] A. Trask, F. Hill, S. Reed, J. Rae, C. Dyer, and P. Blunsom, “Neural arithmetic logic units,” 2018.

[2] B. M. Lake and M. Baroni, “Still not systematic after all these years: On the compositional skills of sequence-to-sequence recurrent networks,” CoRR, vol. abs/1711.00350, 2017.

[3] C. Chiu, T. N. Sainath, Y. Wu, R. Prabhavalkar, P. Nguyen, Z. Chen, A. Kannan, R. J. Weiss, K. Rao, K. Gonina, N. Jaitly, B. Li, J. Chorowski, and M. Bacchiani, “State-of-the-art speech recognition with sequence-to-sequence models,” CoRR, vol. abs/1712.01769, 2017.

[4] Z. Zhang, J. T. Geiger, J. Pohjalainen, A. E. Mousa, and B. W. Schuller, “Deep learning for environmentally robust speech recognition: An overview of recent developments,” CoRR, vol. abs/1705.10874, 2017.

[5] J. Redmon, S. K. Divvala, R. B. Girshick, and A. Farhadi, “You only look once: Unified, real-time object detection,” CoRR, vol. abs/1506.02640, 2015.

[6] S. Ren, K. He, R. B. Girshick, and J. Sun, “Faster R-CNN: towards real-time object detection with region proposal networks,” CoRR, vol. abs/1506.01497, 2015.

[7] T. Lin, P. Goyal, R. B. Girshick, K. He, and P. Dollár, “Focal loss for dense object detection,” CoRR, vol. abs/1708.02002, 2017.

[8] T. Young, D. Hazarika, S. Poria, and E. Cambria, “Recent trends in deep learning based natural language processing,” CoRR, vol. abs/1708.02709, 2017.

[9] G. F. Marcus, “The algebraic mindd: Integrating connectionism and cognitive science,” 2003.

[10] B. M. Lake, T. D. Ullman, J. B. Tenenbaum, and S. J. Gershman, “Building machines that learn and think like people,” CoRR, vol. abs/1604.00289, 2016.

[11] A. Liska, G. Kruszewski, and M. Baroni, “Memorize or generalize? searching for a compositional RNN in a haystack,” CoRR, vol. abs/1802.06467, 2018.

[12] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, “Curriculum learning.”

[13] W. Zaremba and I. Sutskever, “Learning to execute,” CoRR, vol. abs/1410.4615, 2014.

[14] G. Martius and C. H. Lampert, “Extrapolation and learning equations,” CoRR, vol. abs/1610.02995, 2016.

[15] S. S. Sahoo, C. H. Lampert, and G. Martius, “Learning equations for extrapolation and control,” 2018.

[16] E. Jang, S. Gu, and B. Poole, “Categorical Reparameterization with Gumbel-Softmax,” arXiv e-prints, p. arXiv:1611.01144, Nov. 2016.

[17] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, “Automatic differentiation in pytorch,” in NIPS-W, 2017.

The Appendix is structured as follows. In Appendix A, the theoretical justification for the potential improvements are described. This includes a description of the Gumbel softmax (Appendix A.1), bias extension (Appendix A.2), and input independent gate function (Appendix A.3). In Appendix B, we provide an detailed analysis on the instability of the NALU for division. Following, a more elaborate description of the experimental setup is provided in Appendix C. Finally, in Appendix D we provide more detailed results and analysis of these.

A. THEORY

A.1. Gumbel Softmax

The Gumbel Softmax Trick is a modification of the softmax function utilizing the Gumbel distribution, in order to allow for continuous and differentiable reparametrization of discrete random variables, presented in [16].
The first part of the trick consists of reparametrization. A standard Gumbel-distributed variable G is defined using the uniform distribution as

$$G = -\log(-\log(U)),\ U \sim U(0,1). \tag{6}$$

If we then have a discrete random variable X with some distribution $P(X)$ and a random variable α_k such that $P(X = k) \propto \alpha_k$, we can then reparametrize X as:

$$X = \arg\max_k(\log \alpha_k + G_k), \tag{7}$$

where $\{G_k\}_{k \in K}$ are i.i.d. standard Gumbel-distributed variables.
The next step of the trick is then to relax the discrete constraints. If we then consider X as a one-hot vector, we can then use softmax with a temperature term, defined as

$$f_\tau(x)_k = \frac{\exp(\frac{x_k}{\tau})}{\sum_{j=1}^K \exp(\frac{x_j}{\tau})}, \tag{8}$$

yielding the following expression of X^τ

$$X^\tau = f_\tau(\log \alpha + G) = \frac{\exp(\frac{\log \alpha_k + G_k}{\tau})}{\sum_{j=1}^K \exp(\frac{\log \alpha_j + G_j}{\tau})}. \tag{9}$$

The wonderful property of this reparametrization and relaxation is the fact that if we consider the distribution of X^τ using the parameters α, τ, its density has a closed form solution, which can be written as

$$p_{\alpha,\tau}(x) = (n-1)!\tau^{n-1} \prod_{k=1}^K \left(\frac{\alpha_k x_k^{-\tau-1}}{\sum_{j=1}^K \alpha_i x_i^{-\tau-1}} \right), \tag{10}$$

where x is a one-hot vector.

Since the NALU is not defined using any discrete variables, but seeks to emulate discrete choices, directly employing the Gumbel trick does not seem to yield any benefits. If we wish to improve the learning process, we can however introduce the temperature term either in the definition of $\mathbf{g}$ or $\mathbf{W}$, in the following fashion:

$$\mathbf{g}_\tau = \sigma(\tau \mathbf{G}\mathbf{X}),\ \tau \geq 0, \tag{11}$$

or

$$\mathbf{W}_\tau = \tanh(\tau \hat{\mathbf{W}}) \odot \sigma(\tau \hat{\mathbf{M}}), \tag{12}$$

thus changing the optimization surface - by either relaxing or constraining it. We can illustrate the effect of this, by plotting the optimization surfaces of $\mathbf{W}$ and $\mathbf{g}$, for different values of τ

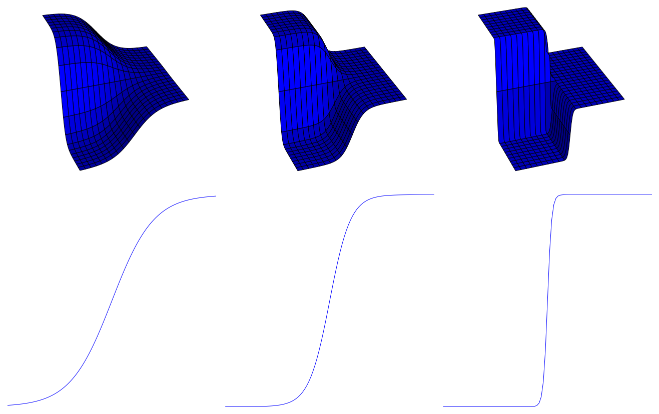

Fig. 5: The optimization surfaces $\mathbf{W}$ and $\mathbf{g}$ for $\tau = \{0.1, 1, 5\}$ from left to right.

As we see in Figure 5, we can greatly modify the optimization surface, relaxing or constraining it by varying τ. It is clear that the learning task over the relaxed surface, low τ, is easier, as it allows for easier movement of the weights, but at the cost of not being nearly as discrete. The opposite is then the case over the constrained surface, high τ. As such, one could believe that by starting training at a low value of τ and then increasing it as it trains, one could prevent early lock-ins of the weights and then end as they become highly discretized.

A.2. Bias Extension

In the definition of Simple-/Complex-Selector of the NALU, $\mathbf{g}$, we see some possible source of trouble in the extrapolation task. Let us for the sake of simplicity consider the two-dimensional case, where $\mathbf{x} = [x_1, x_2]^T$ and $\mathbf{G} = [g_1, g_2]$ and we wish to learn the simple addition operation $\mathbf{Y} = x_1 + x_2$.

To learn this operation, we must then have that our Simple-/Complex selector chooses $\mathbf{g} \approx 1$. Since $\sigma(x) \approx 1, x \geq 5$, then we must have that

$$1 \approx \mathbf{g} = \sigma(\mathbf{G}\mathbf{x}) \Rightarrow 5 \geq \mathbf{G}\mathbf{x} = g_1 x_1 + g_2 x_2, \tag{13}$$

From this, however, it is clear that there exists no numbers g_1, g_2 such that this can hold for $x_1, x_2 \in \mathbb{R}$. If there exists no solution, then by necessity any learned model is not generalizable and will not be able to extrapolate, no matter how well the weights $\mathbf{W}$ are learned.
For a simpler task, such as when $x_1, x_2 \in \mathbb{R}^+$, we can isolate g_1, yielding

$$g_2 > \frac{5 - g_1 x_1}{x_2},\ x_1, x_2 \in \mathbb{R}^+,\ g_1 \in \mathbb{R}, \tag{14}$$

from which we see that as long as x_1 or x_2 are large, it is easy to find g_1, g_2 such that this will hold, but there still does not exist one fixed solution. This means that the simplified model will have an easier time at extrapolating to $x_1, x_2 >> 5$, but will not hold for all $x_1, x_2 \in \mathbb{R}^+$.

To combat this, one can consider including a bias term β. For the same task as before, $x_1, x_2 \in \mathbb{R}$, the problem is now

$$5 \geq g_1 x_1 + g_2 x_2 + \boldsymbol{\beta} \tag{15}$$

which has the simple solution $g_1 = g_2 = 0, \boldsymbol{\beta} > 5$. As such, by including a bias-term, the model should be able to generalize to all values of $x_1, x_2 \in \mathbb{R}$.

A.3. Input-independent gating function

Since for the Simple-/Complex-selector extended with a bias term we wish to learn that the weights g_1, g_2 must equal zero and for β to be the parameter which actually does the choosing between the Simple and the Complex NAC, it then seems apparent to simply do away with the $\mathbf{Gx}$-term. We can then instead simply redefine $\mathbf{g}$ as

$$\mathbf{g}_\beta = \sigma(\boldsymbol{\beta}), \tag{16}$$

to avoid many of the local minima found in the other definition. This could be a promising alteration to the NALU model, if it turns out that the practical training yields the results we expect from the above investigations.

B. NALU DIVISION STABILITY

An important insight in the definition of the complex NAC, equation 3, is the fact that it as a standalone layer cannot learn multiplication and division ($a \times b$, $\frac{a}{b}$) when the signs of a, b are opposite. Since $\exp(x) \geq 0, \forall x$, as well as $\mathbf{g} \geq 0$ by the definition of sigmoid, there exists no way for $\mathbf{m}$ to become negative. This could reasonably simply be remedied by including another connection which learns the sign of the output.

As shown in [1], the NALU seems to have large difficulties in learning division. Since this issue seems isolated to division, one could suspect that it stems from the definition of the Complex NAC instead of the learning process. We wish to investigate the instability of the complex NAC in performing division, by varying the numerator and the denominator for a simple division task, with pre-chosen weights $\mathbf{W} = [1, -1]^T$. The value of ϵ is chosen as the machine-epsilon for 64-bit doubles, since this is the standard representation of floating point values in Python.

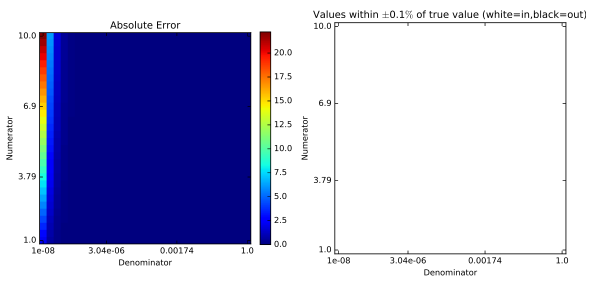

Fig. 6: $\epsilon = 2^{-53} \approx 1.1 \cdot 10^{-16}$, chosen as the machine-epsilon. Keep in mind the logarithmic scale of horizontal axis.

We find here that the larger the numerator is and the closer the denominator is to zero, the larger the absolute error is. This error, however, is within a 0.1% margin of the actual true value. As such, this does not seem to be the actual cause of the instability.

Secondly, we wish to test the performance of the Complex NAC when we vary both ϵ and the denominator, keeping the numerator at 10. We then vary ϵ from the machine-epsilon to values further from zero and the denominator from values close to the machine-epsilon up to one.

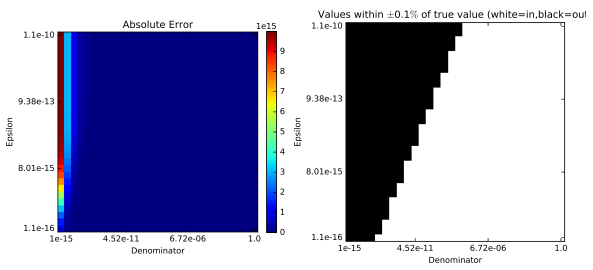

Fig. 7: Numerator: 10. Keep in mind the logarithmic scale of both axes.

We find here that as the denominator approaches the magnitude of ϵ, the absolute error increases drastically. In this case we do find that many of the calculated values are outside the 0.1% margin of the actual true value. As we can see from the image on the right, it is exactly in the cases where ϵ and the numerator are fairly close to each other in magnitude (or the magnitude of ϵ is greater), that division fails.

In general, it would be foolish to choose ϵ as any other value than the machine-epsilon for the system in which NALU is implemented, thus indicating that it is the value of the denominator or the weights for the denominator which must be controlled.

If we consider that the values for the denominator are sampled from a normal distribution $\mathcal{N}(0, 5^2)$, then the probability that choosing a denominator within half the order of magnitude of the machine-epsilon ($1.1 \cdot 10^{-16}$) is just $\int_{-10^{-8}}^{10^{-8}} \frac{1}{\sqrt{2\pi 5^2}} \exp\left(\frac{-x^2}{2 \cdot 5^2}\right) \mathrm{d}x \approx 1.6 \cdot 10^{-9}$, illustrating that the sampling of values for training and validation would likely not be the cause of the failure, but that the failure instead stems from the training process.

In summation, it does not seem like the challenges reported by Trask in extrapolating for the division task stems from the actual definition of the NALU model, but must stem from the learning process, wherein the model either does not manage to learn or learns weights to be so small (and nonzero) that we reach values close to the value of ϵ.

C. EXPERIMENT

In Full Task, we stack two NALUs with sizes respectively (100×2) and (2×1). We also perform similar experiment with two stacked (and similarily shaped) NACs, and a two layer Multi Layer Perceptrons (MLPs) with shape $(100, 50, 50, 1)$. The MLP is chosen as a baseline model. In the Subset Selection Task and the Arithmetic Task, we use a single NALU and single NAC of size (100×2) and (2×1), respectively. In these experiments we use a MLP of size $(100, 50, 2)$ and $(2, 50, 1)$.

The models are implemented in PyTorch 0.4[17] using Python 3.5[5]. All our experiments can be found in our GitHub repository `https://github.com/FrederikWarburg/latent_disagreement`.

D. RESULTS

D.1. Experiment 2: Hyper parameter optimization

We tried optimizing several hyper parameters in Experiment 1 (Section 5.1. We have included the results from some of the more common parameters such as batch size and learning rate. We also performed experiments with alternative optimizer (Adam and SDG), but have chosen not to include the results from these experiments, since neither of the improve the performance and since Trask et al. explicit recommend RMSProp[1]. We also include experiments where we vary the number of unique sets in the training data.

D.1.1. Batch size

We experimented with 4 different batch sizes, and test both interpolation and extrapolation capabilities (Table 8 and Table 9). We expect the increased batch size to smooth the gradients, which could avoid local minima. However, from the tables, it is seen that neither interpolation nor extrapolation are improved by increasing the batch size.

operation	bs = 1	bs = 2	bs = 4	bs = 8
$+$	0.02	32.35	513.52	1675.57
$-$	0.49	5.55	26.93	140.44
$*$	14.66	5153.62	12498.16	2160.07
$/$	0.00	0.02	0.34	10.37

Tab. 8: Interpolation: MSE medianed over 5 experiments. We use the median as opposed the average, as the median is more robust towards outliers.

[5]https://www.python.org/

operation	bs = 1	bs = 2	bs = 4	bs = 8
$+$	inf	inf	inf	inf
$-$	inf	inf	inf	inf
$*$	inf	inf	inf	inf
$/$	25.53	69.07	109.20	3585.39

Tab. 9: Extrapolation: MSE medianed over 5 experiments ignoring nan. We report inf, if the MSE exceeded 10^6.

D.1.2. Learning rate

We further experimented with the learning rate. Trask et al. suggest a learning rate of 0.01 for RMSProp[1], which we alter in several experiments. We report the results for interpolation and extrapolation in Table 10 and 11. Again, we expect that a lower learning rate might help us avoid local minimas and more consistently find the global minima. From these Tables, it is seen that especially the experiments with multiplication gives a high MSE for interpolation. We see that adjusting the learning rate does not seem to improve neither interpolation nor extrapolation, as the best results are obtain with a learning rate of 0.01. As in the previous experiments, it is seen from the extrapolation, that the underlying structure of the operations and the selections, have not been learned.

Operation	lr = 0.1	lr = 0.01	lr = 0.001	lr = 0.0001
$+$	0.03	0.00	0.07	0.01
$-$	0.00	0.00	0.01	0.09
$*$	78.67	2.11	2.71	124.70
$/$	0.04	0.00	0.00	0.00

Tab. 10: Interpolation: MSE medianed over 5 experiments ignoring nan.

Operation	lr = 0.1	lr = 0.01	lr = 0.001	bs = 0.0001
$+$	inf	3108.51	inf	inf
$-$	inf	5.38	inf	inf
$*$	inf	inf	inf	inf
$/$	2581.96	39957.30	53.88	100.16

Tab. 11: Extrapolation: MSE medianed over 5 experiments ignoring nan. We report inf, if the MSE exceeded 10^5.

These experiments were replicated for different optimizers. We tried both SDG and Adam, however, neither improved the performance. Thus, we have not included the results in the report.

D.1.3. Set size

As we expected the set size to be too small, we investigated how the size of the training set affected the loss. We tried three different set sizes, and the results are reported in Table 12 and 13. Based on these tables, it does not seem that neither interpolation nor extrapolation capabilities improves by increasing the training size.

Operation	set size = 100	set size = 1000	set size = 10000
$+$	802.58	1033.25	2146.30
$-$	346.56	349.23	18.48
$*$	1116.73	35299.51	12387.78
$/$	0.26	0.02	0.01

Tab. 12: Interpolation: MSE medianed over 5 experiments ignoring nan.

operation	set size = 100	set size = 1000	set size = 10000
$+$	inf	inf	inf
$-$	inf	inf	inf
$*$	inf	inf	inf
$/$	17.17	42.09	inf

Tab. 13: Extrapolation: MSE medianed over 5 experiments ignoring nan. We report inf, if the MSE exceeded 10^5.

D.2. Simple function learning

The experiments on the Arithmetic Task were conducted using two random input numbers per iteration in a given range. Thus, the size of $\mathbf{W}$ becomes 2×1 and $\mathbf{g}$ is 1×1 and $\mathbf{G}$ becomes 2×1. In each experiment, we use different input ranges and observe the convergence of the weights and losses when trying to learn the addition operation. For all experiments we expect to learn $\mathbf{W} = [1\,,\,1]$, $g = 1$, meaning that we choose the simple NAC gate and simply add the elements together.

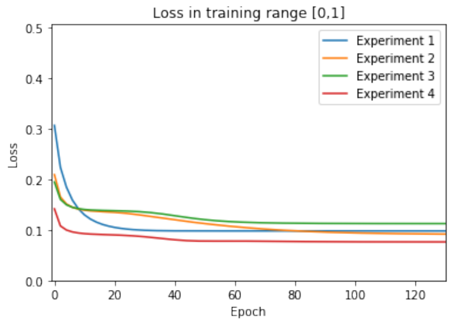

Fig. 8: Training loss

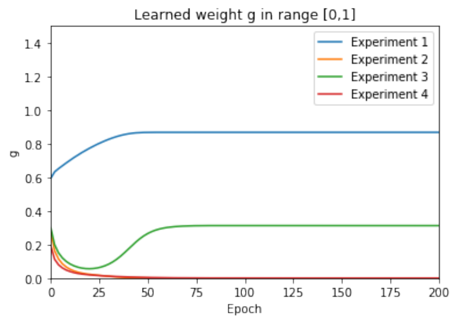

Fig. 9: Learned gate

The training loss clearly indicates that each of the models are able to converge smoothly after around 100 epochs, with limited change beyond this. When looking at the learned gates, we see a wide range of different values for g in each of the experiments, despite the fact that each of them is identical. This shows that there is some clear instability when learning this gating function, and in some of the cases the gate is chosen to be a weighting between the simple and complex NAC. This evidently means that the model will have trouble extrapolating beyond the training range, as this results in multiplication between a and b. This points towards the fact that the gating function has a hard time converging towards the desired value. A similar experiment is conducted with the NALU trained in the range $[-1, 1]$. This is done in order to investigate if values from both the positive and negative range will improve training, hopefully making the NALU choose the simple NAC. In Figure 10 we now see that each of the 4 experiments seem to converge during training, however the learned gate is still not learned as the simple NAC.

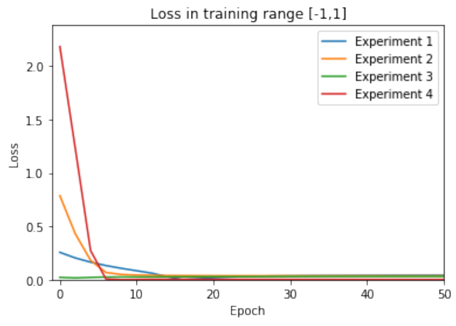

Fig. 10: Training loss

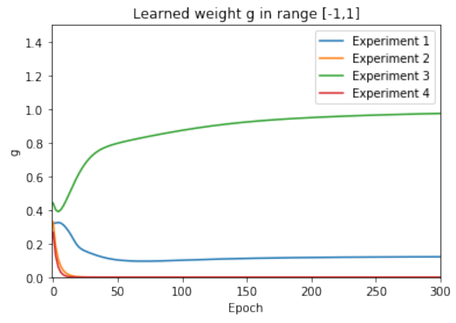

Fig. 11: Learned gate

Learning the gate generally seems to be a problem when trying to training this type of network, and the results in Table 2 also shows the lack of generalization. As such, it seems futile to extend the experiment to other types of arithmetic operations, as the problem clearly lies in the gating function, and this problem is likely to persist with different types of operations.

D.3. Weight Space Analysis

This subsection concerns the task of learning subset selection. For a series of NALU experiments, there are not consistent convergence across the experiments, with some getting stuck during learning and simply not finding any connection between the input and output. This is most likely stemming from the fact that the gate $\mathbf{g}$ is not learned properly, resulting in the use of the complex NAC unit, and eventually leading to a multiplication of the elements in **x**. As such, the loss explodes very quickly, and it seems hard to recover from here.

In Figure 12 we look at the weight matrix **W** in order to see what happens to the weights during training. Ideally, we want some of the elements to go towards 1 (red), indicating that the corresponding elements in the input vector **x** should be included in the subset selection. Other elements must go to 0 (cyan) to exclude them from the selection.

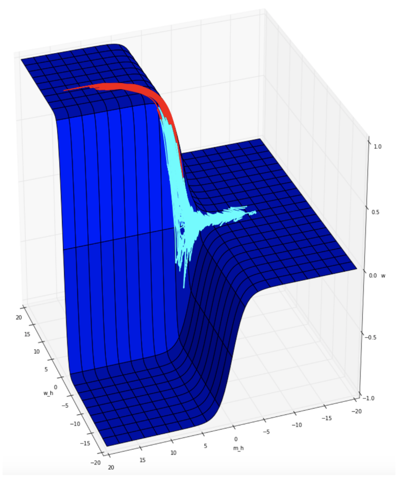

Fig. 12: Propagation of the learned weights. 200 lines in the plot corresponding to the elements of **W** plotted over time. Red lines are the elements that should converge to 1 (indicating selection). The cyan lines should converge to 0 - indicating that their respective elements in **x** should not be selected. From the Figure we see the expected convergence.

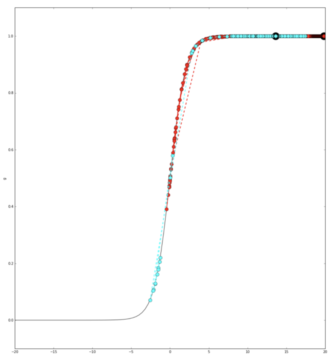

Fig. 13: Propagation of the learned gating function. The red and the cyan points illustrate the time-steps for the convergence of the two elements in **g**. Both elements end at the expected value of 1 (indicated by a black circle).

In the case here, the weights corresponding to the included input elements successfully go to 1, whereas the elements to be excluded eventually find their way to 0. When looking at the gating function **g** in Figure 13 we also see how the network seems to learn that it must choose the simple NAC in order to add the elements together.

This was however not the case in most of the experiments, as it seemed very unstable in learning the right gating function. An example is shown in Figure 14, where the weights cannot learn the underlying structure. These results seem to arise from the fact that the gating function gets stuck at 0, meaning that the network evidently chooses the complex NAC (Figure 15).

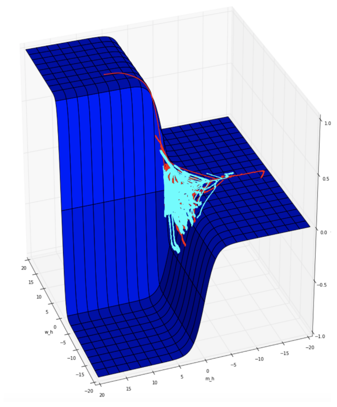

Fig. 14: Propagation of the learned weights. We see 200 lines in the plot which correspond to the elements of **W** plotted over time. The red lines are the elements that we expect to converge to 1 - indicating that these elements of **X** should be selected - and the cyan lines, we expect to converge to 0 - indicating that their respective elements in **X** should not be selected. From the Figure we do not see the expected convergence as both the red and cyan weights get stuck on the zero-plane.

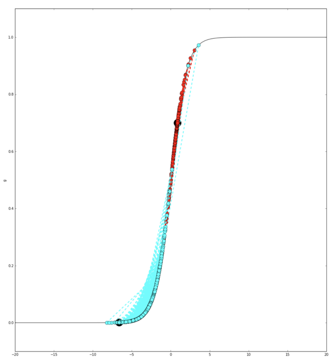

Fig. 15: Propagation of the learned gating function. The red and the cyan points illustrate the time-steps for the convergence of the two elements in **g**. We expect both elements end at 1 (indicated by a black circle), however, the cyan weight ends at the zero plane - thus choosing the complex NAC.

It is very interesting that we do not actually observe **g** getting stuck in either the 0-plane or the 1-plan, but instead keeps fiddling around in areas which go from low to high gradient. This indicates that a possible way to fix this, is to decrease the acceleration of the graph in this area, by smoothing e.g. with a temperature term.

D.4. Ones-initialization

From Table 3, we see that the Ones-initialization is the only initialization with a 100%-convergence rate. If we consider that $x_i \in U(0,1)$ and $\mathbf{G} = \mathbf{1}$, then the initial $\mathbf{g} = \sigma(\mathbf{G}\mathbf{x}) \approx \sigma([100 \cdot 0.5, 100 \cdot 0.5]^T) = \sigma([50, 50]^T) = [1, 1]^T$. This then means that the Simple-/Complex-selector $\mathbf{g}$ is initialized firmly stuck in a flat plane to select Simple and unlikely to ever exit that plane. This turns out to be well-suited when we do want to only select the Simple NAC in the NALU, but absolutely terrible for any other situations. The other initializations are sampled from zero-mean distributions, which does not have this problem and instead initializes $\mathbf{g} \approx \sigma([0, 0]^T) = [0.5, 0.5]^T$, which would be in the very center of the optimization space.