Neural Power Units

Niklas Heim, Tomáš Pevný, Václav Šmídl
Artificial Intelligence Center
Czech Technical University
Prague, CZ 120 00
`{niklas.heim, vasek.smidl, tomas.pevny}@aic.fel.cvut.cz`

Abstract

Conventional Neural Networks can approximate simple arithmetic operations, but fail to generalize beyond the range of numbers that were seen during training. *Neural Arithmetic Units* aim to overcome this difficulty, but current arithmetic units are either limited to operate on positive numbers or can only represent a subset of arithmetic operations. We introduce the *Neural Power Unit* (NPU) that operates on the *full domain of real numbers* and is capable of *learning arbitrary power functions* in a single layer. The NPU thus fixes the shortcomings of existing arithmetic units and extends their expressivity. This is achieved by internally using complex weights without requiring a conversion of the remaining network to complex numbers. We show where the NPU outperforms its competitors in terms of accuracy and sparsity on artificial arithmetic datasets. Additionally, we demonstrate how the NPU can be used as a highly interpretable model to discover the generating equation of a dynamical system purely from data.

1 Introduction

Numbers and simple algebra are essential not only to human intelligence but also to the survival of many other species [Dehaene, 2011, Gallistel, 2018]. A successful, intelligent agent should therefore be able to perform simple arithmetic. State of the art neural networks are capable of learning arithmetic, but they fail to extrapolate beyond the ranges seen during training [Suzgun et al., 2018, Lake and Baroni, 2018]. The inability to generalize to unseen inputs is a fundamental problem that hints at a lack of *understanding* of the given task. The model merely memorizes the seen inputs and fails to abstract the true learning task. The failure of numerical extrapolation on simple arithmetic tasks has been shown by Trask et al. [2018], who also introduce a new class of *Neural Arithmetic Units* that show good extrapolation performance on some arithmetic tasks.

Including Neural Arithmetic Units in common neural networks, promises to significantly increase their extrapolation capabilities due to their inductive bias towards numerical computation. This is especially true for tasks in which the data contains mathematical relationships. Additionally, they promise to reduce the amount of parameters that are needed for a given task, and can therefore drastically improve the explainability of the model. We demonstrate how neural arithmetic units as part of a *Neural Ordinary Differential Equation* (NODE, Chen et al. [2019]) can outperform a dense network with significantly fewer parameters in Sec. 4.1. In fact, our NPU can be used to directly read out the correct generating ODE from the fitted model. This is inline with recent efforts to build *interpretable* models instead of *explaining* black box models [Rudin, 2019], like conventional neural networks.

The currently available arithmetic units all have different strengths and weaknesses, but none of them solve simple arithmetic completely. The *Neural Arithmetic Logic Unit* (NALU) by Trask et al. [2018], chronologically, was the first arithmetic unit. It can solve addition ($+$, including subtraction),

Preprint. Under review.

multiplication ($\times$), and division ($\div$), but is limited to positive inputs. The convergence of the NALU is quite fragile due to an internal gating mechanism between addition and multiplication paths as well as the use of a logarithm which is problematic for small inputs. Recently, Schlör et al. [2020] introduced the *improved NALU* (iNALU, to fix the NALU's shortcomings. However, it significantly increases its complexity and we observe only a slight improvement in performance.
Madsen and Johansen [2020] solve ($+, \times$) with two new units: the *Neural Addition Unit* (NAU), and the *Neural Multiplication Unit* (NMU). Instead of gating between addition and multiplication paths, they are separate units that can be stacked. They can work with the full range real numbers, converge much more reliably, but are not capable of representing division.

Our Contributions

Neural Power Unit. We introduce a new arithmetic layer (NPU, Sec. 3) which is capable of learning power functions (x^w) including multiplication ($x \times y = x^1 y^1$) and division ($x \div y = x^1 y^{-1}$). Stacks of NAUs and NPUs can thus learn the full spectrum of simple arithmetic operations.

A fix for NALU. We isolate the NALU's multiplication path and lift it into complex space, which enables correct processing of negative inputs (Sec. 3.1). Still, it is possible to include the NPU in a larger, real valued network without a conversion of the original model to complex numbers. Further, we improve upon the known convergence issues of the NALU by introducing an *relevance gate* that smoothes out the loss surface of the NPU (Sec. 3.2). With the relevance gate the NPU reaches extrapolation and sparsity errors that are on par with the NMU on ($\times$) and outperform NALU on ($\div, \sqrt{\cdot}$).

Interpretability. we show how the NPU can be used as a highly interpretable model for equation discovery of the fractional SIR model (Sec. 4.1) that was used to fit the COVID-19 outbreak in various countries [Taghvaei et al., 2020].

2 Related Work

A number of different approaches to automatically solve arithmetic tasks have been studied in the recent years. Approaches include Neural GPUs [Kaiser and Sutskever, 2016], Grid LSTMs [Kalchbrenner et al., 2016], Neural Turing Machines [Graves et al., 2014], and Neural Random Access Machines [Kurach et al., 2016]. They solve tasks like binary addition and multiplication, or single digit arithmetic. The Neural Status Register [Faber and Wattenhofer, 2020] focusses on control flow. The Neural Arithmetic Expression Calculator [Chen et al., 2018], a hierarchical reinforcement learner is the only method that solves the division problem, but it operates on character sequences of arithmetic expressions. Related is symbolic integration with transformers [Lample and Charton, 2019]. Unfortunately, most of the named models have severe problems with extrapolation [Madsen and Johansen, 2019, Saxton et al., 2019]. A solution to the extrapolation problem could be Neural Arithmetic Units. They are designed with an inductive bias towards systematic, arithmetic computation. However, currently they are limited in their capabilities of expressing the full range of simple arithmetic operations ($+, \times, \div$). In the following two sections we briefly describe the currently available arithmetic layers including their advantages and drawbacks.

2.1 Neural Arithmetic Logic Units

Trask et al. [2018] have demonstrated the severity of the extrapolation problem of dense networks for even the simplest arithmetic operations, such as summing or multiplying two numbers. In order to increase the power of abstraction to for arithmetic tasks they propose the *Neural Arithmetic Logic Unit* (NALU) which is capable of learning ($+, \times, \div$). However, the NALU cannot handle negative inputs correctly due to the logarithm in Eq. 2:

Definition (NALU). *The NALU consits of a ($+$) and a ($\times$) path that share their weights $\hat{\boldsymbol{W}}$ and $\hat{\boldsymbol{M}}$.*

$$\textit{Addition: } \boldsymbol{a} = \boldsymbol{W}\boldsymbol{x} \qquad \boldsymbol{W} = \tanh(\hat{\boldsymbol{W}}) \odot \sigma(\hat{\boldsymbol{M}}) \tag{1}$$

$$\textit{Multiplication: } \boldsymbol{m} = \exp \boldsymbol{W}(\log(|\boldsymbol{x}| + \epsilon)) \tag{2}$$

$$\textit{Output: } \boldsymbol{y} = \boldsymbol{a} \odot \boldsymbol{g} + \boldsymbol{m} \odot (1 - \boldsymbol{g}) \qquad \boldsymbol{g} = \sigma(\boldsymbol{G}\boldsymbol{x}) \tag{3}$$

Additionally, the logarithm destabilizes training to an extent that the chance of success can drop below 20% for ($+, \times$), it becomes practically impossible to learn ($\div$), and difficult to learn from small inputs in general [Madsen and Johansen, 2019]. Schlör et al. [2020] provide a detailed description of the shortcomings of the NALU and they suggest an *improved NALU* (iNALU). The iNALU addresses the NALU's problems through a number of mechanisms. It has independent addition and multiplication weights for Eq. 1 and Eq. 2, clips weights and gradients to improve training stability, regularizes the weights to push them away from zero, and, most importantly, introduces a mechanism to recover the sign that is lost due to the absolute value in the logarithm. Additionally, the authors propose to reinitialize the network if its loss is not improving during training. We include the iNALU in one of our experiments and find that it only slightly improves the NALU's performance (Sec. 4.2) at the cost of a significantly more complicated unit. Our NPU avoids all these mechanisms by internally using complex numbers.

2.2 Neural Multiplication Unit & Neural Addition Unit

Instead of trying to fix the NALU's convergence issues Madsen and Johansen [2020] propose a new unit for $(\times)$ only. The *Neural Multiplication Unit* (NMU) uses explicit multiplications and learns to gate between identity and $(\times)$ of inputs. The NMU is defined by Eq. 4 and is typically used in conjunction with the so-called *Neural Addition Unit* (NAU) in Eq. 5.

Definition (NMU & NAU). *NMU and NAU are two units that can be stacked to model $(+, \times)$.*

$$\text{NMU: } y_j = \prod_i M_{ij} z_i + 1 - M_{ij} \qquad M_{ij} = \min(\max(\tilde{M}_{ij}, 0), 1) \tag{4}$$

$$\text{NAU: } \boldsymbol{y} = \boldsymbol{A}\boldsymbol{x} \qquad A_{ij} = \min(\max(\tilde{A}_{ij}, -1), 1) \tag{5}$$

Both NMU and NAU are regularized with $\mathcal{R} = \sum_{ij} \min(|W_{ij}|, |1 - W_{ij}|)$ and their weights are clipped, which biases them towards learning an operation or pruning it completely. The combination of NAU and NMU can thus learn $(+, \times)$ for both positive and negative inputs. Training NAU and NMU is stable and succeeds much more frequently than with the NALU, but they cannot represent $(\div)$, which we address with our NPU.

3 Neural Power Units

To fix the deficiencies of current arithmetic units, we propose a new arithmetic unit (inspired by NALU) that can learn arbitrary power functions (x^w) (including $\times, \div$) for positive and negative numbers, and still train well. Combined with the NAU we solve the full range of arithmetic operations. This is possible through a simple modification of the $(\times)$-path of the NALU (Eq. 6). We suggest to use the complex logarithm and to allow $\boldsymbol{W}$ to become complex. The complex logarithm is also defined for negative inputs and a complex $\boldsymbol{W}$ aides convergence (see Sec. 4.1). The improvement during training might be explained by the additional imaginary parameters that make it possible to avoid regions with an uninformative gradient signal.

3.1 Naive Neural Power Unit – NaiveNPU

With the modifications introduced above we can extend the multiplication path of the NALU from

$$\boldsymbol{m} = \exp \boldsymbol{W}(\log_{\text{real}}(|\boldsymbol{x}| + \epsilon)) \tag{6}$$

to use the complex logarithm ($\log \coloneqq \log_{\text{complex}}$) and a complex weight $\boldsymbol{W}$ to

$$\boldsymbol{z} = \exp(\boldsymbol{W} \log \boldsymbol{x}) = \exp\left((\boldsymbol{W_r} + i\boldsymbol{W}_i) \log \boldsymbol{x}\right). \tag{7}$$

The complex log in Eq. 7 lifts the positivity constraint on $\boldsymbol{x}$ resulting in a layer that can process both positive and negative numbers correctly. A complex weight matrix $\boldsymbol{W}$ somewhere in a larger network would result in complex gradients in other layers. This would effectively result in doubling the number of parameters of the whole network. As we are only interested in real networks outputs, we can avoid this doubling by considering only the real part of the output $\boldsymbol{z}$:

$$\begin{aligned} \text{Re}(\boldsymbol{z}) &= \text{Re}(\exp((\boldsymbol{W}_r + i\boldsymbol{W}_i)(\log \boldsymbol{r} + i\pi\boldsymbol{k}))) & (8)\\ &= \exp(\boldsymbol{W}_r \log \boldsymbol{r} - \pi\boldsymbol{W}_i\boldsymbol{k}) \odot \cos(\boldsymbol{W}_i \log \boldsymbol{r} + \pi\boldsymbol{W}_r\boldsymbol{k}). & (9)\end{aligned}$$

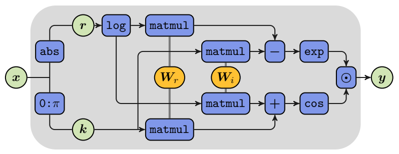

Figure 1: NaiveNPU diagram, with input $\boldsymbol{x}$ and output $\boldsymbol{y}$. Vectors in green, trainables in orange, functions in blue.

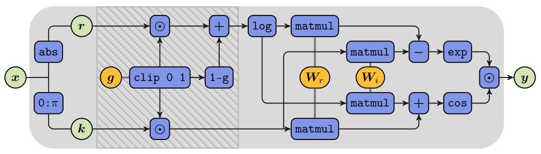

Figure 2: NPU diagram. The NPU has a relevance gate $\boldsymbol{g}$ (hatched background) in front of the input to the actual unit to prevent vanishing gradients.

Above we have used Euler's formula[1] and the fact that the complex logarithm for real valued inputs is

$$\log x = \log r + i\theta = \log r + ik\pi, \tag{10}$$

where $k = 0$ if $r \geq 0$ and $k = 1$ if $r < 0$. A diagram of the NaiveNPU is shown in Fig. 1.

Definition (NaiveNPU). *The Naive Neural Power Unit with real and imaginary matrices $\boldsymbol{W}_r$ and $\boldsymbol{W}_i$ is defined as*

$$\boldsymbol{z} = \exp(\boldsymbol{W}_r \log \boldsymbol{r} - \boldsymbol{W}_i \boldsymbol{k}) \odot \cos(\boldsymbol{W}_i \log \boldsymbol{r} + \boldsymbol{W}_r \boldsymbol{k}), \text{ where} \tag{11}$$

$$\boldsymbol{r} = |\boldsymbol{x}|, \quad k_i = \begin{cases} 0 & x_i \leq 0 \\ \pi & x_i > 0 \end{cases}$$

3.2 The Relevance Gate – NPU

The NaiveNPU has difficulties to converge on large scale tasks, and to reach sparse results in cases where the input to a given row is small. We demonstrate this on a toy example of learning the identity on one of two inputs (neglecting the second one):

$$\mathcal{L} = |\mathrm{m}(x_1, x_2) - x_1|, \quad \text{where} \quad \mathrm{m} = \mathrm{NPU}(2,1), \quad x_1 \sim \mathcal{U}(0,2), \quad x_2 \sim \mathcal{U}(0, 0.05).$$

The left plot in Fig. 3 depicts the gradient of a batch of two dimensional inputs, one of which is small and irrelevant. It is clearly visible that even for this simple example, a large part of the gradient surface of the NaiveNPU is zero. This vanishing gradient can be explained as follows. One row of NaiveNPU weights effectively raises each input to a power and multiplies them: $x_1^{w_1} x_2^{w_2} \dots x_n^{w_n}$. If a single input x_i is constantly close to zero (i.e. irrelevant), the whole row will be zero, no matter what its weights are and the gradient information on all other weights is lost. We therefore introduce a gate on the input of the NPU that can turn irrelevant inputs into 1s. A diagram of the NPU is shown in Fig. 2.

Definition (NPU). *The NPU extends the NaiveNPU by the relevance gate $\boldsymbol{g}$ on the input $\boldsymbol{x}$.*

$$\boldsymbol{z} = \exp(\boldsymbol{W}_r \log \boldsymbol{r} - \boldsymbol{W}_i \boldsymbol{k}) \odot \cos(\boldsymbol{W}_i \log \boldsymbol{r} + \boldsymbol{W}_r \boldsymbol{k}), \text{ where} \tag{12}$$

$$\boldsymbol{r} = \boldsymbol{g} \odot |\boldsymbol{x}| + (1 - \boldsymbol{g}), \quad k_i = \begin{cases} 0 & x_i \leq 0 \\ g_i\pi & x_i > 0 \end{cases}, \quad g_i = \min(\max(g_i, 0), 1) \tag{13}$$

The central plot of Fig. 3 shows the gradient of the NPU on the identity task with its initial gate setting of $g_1 = g_2 = 0.5$. The large zero-gradient region of the NaiveNPU is gone. The last plot shows the same loss for $g_1 = 1$ and $g_2 = 0$, which corresponds to the correct gates at the end of NPU training. The gradient is independent of w_2, which means that it can easily be pruned by a simple regularization such as L1. In Sec. 4.3 we show how important the relevance gating mechanism is for the convergence and sparsity of large models. Sparsity is especially important in order to use the NPU as an interpretable model.

3.3 Initialization

We recommend to initialize the NPU with a Glorot Uniform distribution on the real weights $\boldsymbol{W}_r$. The imaginary weights $\boldsymbol{W}_i$ can be initialized to zeros, so they will only be used where necessary, and the gate $\boldsymbol{g}$ with 0.5, so the NPU can choose to output 1.

should this be longer? or not a section at all?

[1]Euler's formula states that for any real number x: $e^{ix} = \cos x + i \sin x$

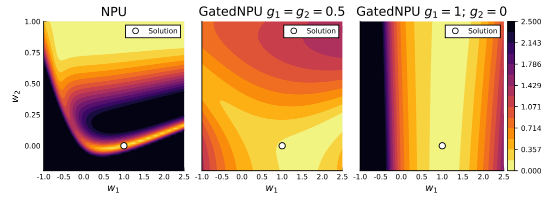

Figure 3: Gradient surfaces of NaiveNPU and NPU for the task of learning the identity on x_1. Inputs and loss are defined on the right, gradient surfaces on the left (black areas are beyond the color scale). The correct solution is $w_1 = 1$ and $w_2 = 0$. The NaiveNPU has a large zero gradient region for $w_2 > 0.75$, while the NPU's surface is much more informative. The gates for central plot are fixed at $g_1 = g_2 = 0.5$ which corresponds to the initial gate parameters. During training they will adjust as needed, in this case to $g_1 = 1$ and $g_2 = 0$. $\boldsymbol{W}_i$ is set to zero in all plots.

4 Experiments

In Sec. 4.1 we show how the NPU can be used as a highly interpretable model which can extract the generating equation of an ODE that contains fractional powers. In Sec. 4.2 & 4.3 we provide experiments on standard arithmetic tasks in which the NPU outperforms the NALU in all tasks.

4.1 Equation Discovery of an Epidemiological Model

other titles: Discovering the Equations of an Epidemic

Data-driven models such as *SINDy* [Champion et al., 2019] or *Neural Ordinary Differential Equations* (NODE, Chen et al. [2019]) are used more and more in scientific applications. Recently, *Universal Differential Equations* (UDEs, Rackauckas et al. [2020]) were introduced which aim to combine data-driven models with physically informed differential equations to maximize explainability of the resulting models. NODEs and UDEs can be tricky to fit, because they run into the same difficulties as RNNs like vanishing gradients and bifurcations. With this experiment we want to demonstrate that the NPU works reliably in such settings, despite the convergence issues that its predecessor (the NALU) has - even for non-sequential tasks. If an ODE contains parameters that cannot easily be modelled by dense layers, the NPU can be a good choice because of its ability to learn any power function. Additionally, if interpretability of the result is important, the NPU can be used recover the generating ODE because of its interpretable design, instead of trying to explain a black box model in retrospect.

An example of an ODE that contains powers is the fractional SIR model (fSIR, Taghvaei et al. [2020]). The classical SIR model is built from three quantities of interest: S (susceptible), I (infectious), and R (recovered/removed). Arguably the most important part of the model is the transmission rate r, which is typically taken to be proportional to the product of S and I. Taghvaei et al. [2020] argue that, especially in the initial phase of an epidemic, the boundary areas of infected and susceptible cells scale with a fractional power, which leads to Eq. 15:

$$\frac{dS}{dt} = -r(t) + \eta R(t), \qquad \frac{dI}{dt} = r(t) - \alpha I(t), \qquad \frac{dR}{dt} = \alpha I(t) - \eta R(t), \tag{14}$$

$$r(t) = \beta I(t)^\gamma S(t)^\kappa, \tag{15}$$

We solve the fSIR model numerically with the parameters $\alpha = 0.05$, $\beta = 0.06$, $\eta = 0.01$, $\gamma = \kappa = 0.5$, and 40 datapoints that are equally spaced in the time interval $T = (0, 200)$. The initial conditions are set to $S_0 = 100$, $I_0 = 0.01$, and $R_0 = 0$.

We fit the fSIR model with three different model types: a dense network, the NPU, and an NPU with imaginary weights fixed to zero (further referred to as *real NPU*). An exemplary model is: $\text{NPU} = \text{Chain}(\text{NPU}(3, h), \text{NAU}(h, 3))$ with variable hidden size h. The detailed models are defined in Tab. A5. The training objective is the loss $\mathcal{L}$ with L1 regularization.

$$\mathcal{L} = \text{MSE}(\boldsymbol{t}, \text{model}(\boldsymbol{u}_0)) + \beta ||\boldsymbol{\theta}||_1. \tag{16}$$

We train each model for 3000 steps with the ADAM optimizer and a learning rate of 0.005, and after that with LBFGS until convergence (or for maximum 1000 steps). For each model type we run a small grid search to build a Pareto front with $h \in \{6, 9, 12, 15, 20\}$ and $\beta \in \{0, 0.01, 0.1, 1\}$, where each hyper-parameter pair is run five times. The resulting Pareto front is shown on the left of Fig. 4.

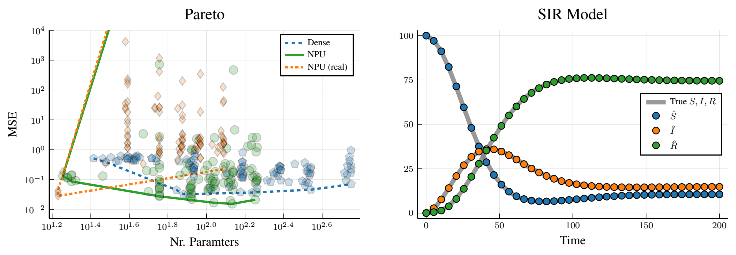

Figure 4: Pareto fronts of the dense network, the NPU, and the NPU with imaginary weights fixed to zero. The NPU reaches solutions with lower MSE and fewer parameters than the dense net. The real NPU mostly yields worse results than the NPU, just in a few cases it converges to very sparse models with good MSE.

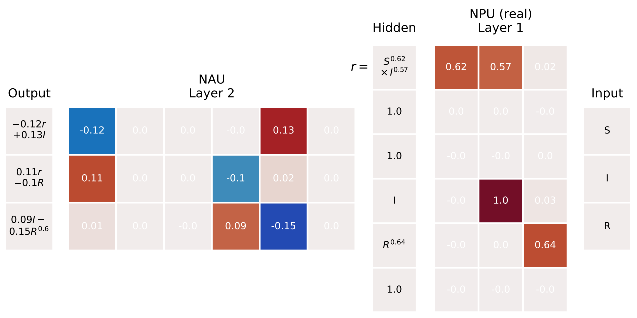

Figure 5: Visualization of the best real NPU. Reading from right to left, it takes the SIR variables as an input, then applies the NPU and the NAU. It correctly identifies r as a fractional product in the NPU, and gets the rest of the fSIR parameters almost right in the NAU.

The NPU clearly reaches much sparser and better solutions than the dense network. The real NPU has problems to converge in the majority of cases. However, there are a few models in the bottom left that reach a very low MSE and have very few parameters. The best these models is shown in Fig. 5. It looks strikingly similar to the fSIR model in matrix from:

$$\begin{bmatrix} \dot{S} \\ \dot{I} \\ \dot{R} \end{bmatrix} = \begin{bmatrix} -\beta & 0 & \eta \\ \beta & -\alpha & 0 \\ 0 & \alpha & \eta \end{bmatrix} \begin{bmatrix} I^\gamma S^\kappa \\ I \\ R \end{bmatrix}. \tag{17}$$

Reading Fig. 5 from right to left, we can extract the ODE that the real NPU represents. It identified the transmission rate correctly as a product of two fractional powers $r = I^\gamma S^\kappa$ with $\kappa = 0.57$ and $\gamma = 0.62$, which is close to the true values $\gamma = \kappa = 0.5$. In the second layer the NAU combines the correct hidden outputs from the NPU such that $\dot{S}$ is composed of the negative transmission rate r and positive R. $\dot{I}$ and $\dot{R}$ are also composed from the correct hidden variables, with the parameters α, β, η being not far off from the truth. We conclude that even with this very naive approach the real NPU is able to recover a fractional SIR model.
In summary, the NPU clearly works well in sequential tasks, and we have shown that we can reach highly interpretable results with the real NPU, but in practice, using the real NPU might be difficult due to its lower success rate. With a more elaborate analysis it might be possible to reach the same solutions with the full NPU and e.g. a strong regularization of its imaginary parameters.

4.2 Simple Arithmetic Task

In this experiment we demonstrate the capabilities of five different layers (NPU, NMU, NALU, iNALU, Dense) on a small problem with two inputs and four outputs. The objective is to learn the

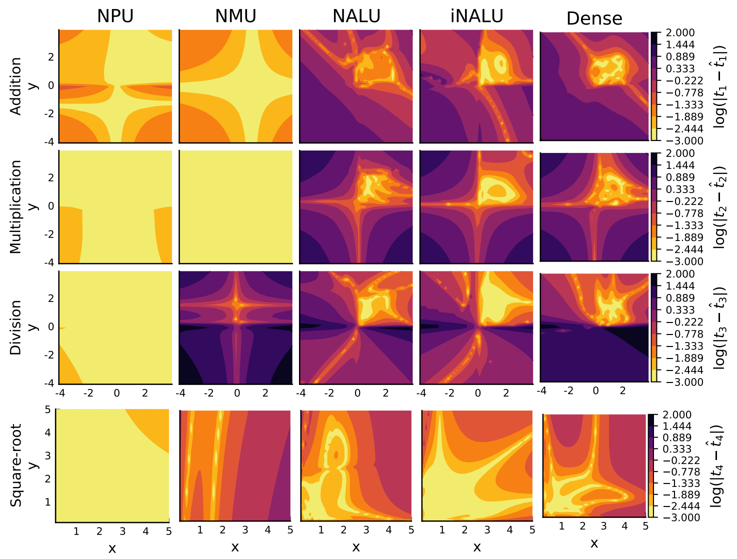

Figure 6: Comparison of different models learning Eq. 18. Each column represents the best model of 20 runs. The dense network learns all tasks, but fails to extrapolate. NALU and iNALU are better at extrapolating to positive numbers, but fail if one of the inputs is negative. The NMU successfully learns $(+, \times)$, but not $(\div, \sqrt{\cdot}\,)$. Only the NPU learns all tasks and can extrapolate both to positive and negative numbers.

function $f : \mathbb{R}^2 \to \mathbb{R}^4$ with a standard MSE loss:

$$f(x, y) = (x + y,\ xy,\ x/y,\ \sqrt{x}\ )^T =: \boldsymbol{t}, \tag{18}$$

$$\mathcal{L} = \frac{1}{4}\sum_{i=1}^{n=4} (\text{model}(x, y)_i - f(x, y)_i) = \text{MSE}(\hat{\boldsymbol{t}}, \boldsymbol{t}). \tag{19}$$

Each model has two layers with a hidden dimension $h = 6$. E.g. the NPU model is defined by $\text{NPU} = \text{Chain}(\text{NPU}(2, 6), \text{NAU}(6, 4))$. The remaining models that are used in the tables and plots are given in Tab. A2. To obtain valid results in case of division we train on positive, non-zero inputs, but test on negative, non-zero numbers (except for test inputs to the square-root):

$$(x_{\text{train}}, y_{\text{train}}) \sim \mathcal{U}(0.1, 2) \quad (x_{\text{test}}, y_{\text{test}}) \sim \mathcal{R}(\text{-4.1:0.2:4}) \quad (x_{\text{test, sqrt}}, y_{\text{test, sqrt}}) \sim \mathcal{R}(\text{0.1:0.1:4}) \tag{20}$$

where $\mathcal{R}$ denotes a *range* with start, step, and end. We train each model for 20 000 steps with the ADAM optimizer, a learning rate of 0.001, and a batch size of 100. The input samples are generated on the fly during training. Fig. 6 shows the error surface of the best of 20 models on each task. Tab. A1 lists the corresponding averaged testing errors of all 20 models.

The NPU successfully learns $(+, \times, \div, \sqrt{\cdot})$ and clearly outperforms NALU and iNALU on all tasks. The NPU is on par with the NMU for $(+)$, but the NMU is better at $(\times)$ due to its inductive bias. The NMU cannot learn $(\div, \sqrt{\cdot})$. The NPU excels at extrapolation on all tasks.

4.3 Large Scale Arithmetic Task

One of the most important properties of a layer in a neural network is its ability to scale. With the large scale arithmetic task we show that the NPU works reliably on many-input tasks that are heavily over-parametrized. In this section we compare NALU, NMU, NPU, and the NaiveNPU on a task that is identical to the 'arithmetic task' that Madsen and Johansen [2020] and Trask et al. [2018] analyse as well. The goal is to sum two subsets of a 100 dimensional vector and apply an operation (like $\times$) to the two summed subsets. The dataset generation is defined in the set of Eq. 21, with the parameters from Tab. A4.

$$a = \sum_{i=s_{1,\text{start}}}^{s_{1,\text{end}}} x_i, \quad b = \sum_{i=s_{2,\text{start}}}^{s_{2,\text{end}}} x_i, \quad t_{\text{add}} = a + b, \quad t_{\text{mul}} = a \times b, \quad t_{\text{div}} = 1/a, \quad t_{\text{sqrt}} = \sqrt{a}, \tag{21}$$

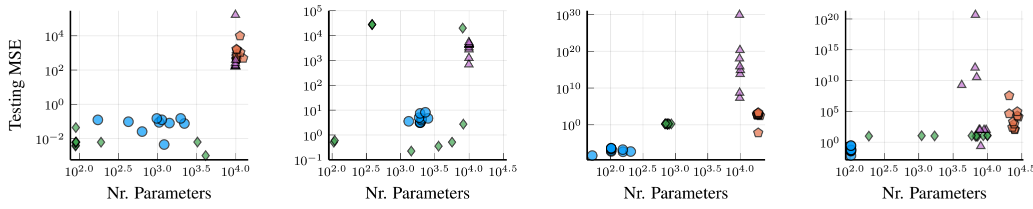

Figure 7: Validation MSE over number of non-zero parameters ($w_i > 0.001$) of the large scale arithmetic task. Again, the NMU outperforms the NPU on its native tasks, addition and multiplication. The NPU is the best at division and square-root. The NaiveNPU without the relevance gate is far off, because it does not have the necessary gradient signal to converge, as discussed in Sec. 3.2

Table 1: Testing errors of the large scale arithmetic task.

Task	NPU	NALU	NMU	NaiveNPU
$+$	0.091 ± 0.048	1700.0 ± 2900.0	**0.009 ± 0.012**	17000.0 ± 52000.0
$\times$	**4.7 ± 1.8**	4.0e94 ± 1.2e95	10000.0 ± 14000.0	3600.0 ± 1500.0
$\div$	**2.0e-7 ± 2.0e-7**	740.0 ± 620.0	1.69 ± 0.24	Inf ± NaN
$\sqrt{\cdot}$	**0.09 ± 0.1**	4.0e6 ± 1.1e7	10.8 ± 1.2	Inf ± NaN

where starting and ending values $s_{i,\text{start}}$, $s_{i,\text{end}}$ of the summations are chosen such that a and b come from subsets of the input vector $\boldsymbol{x}$ with a given overlap. Specifics of the used models are defined in Tab. A3. Madsen and Johansen [2020] perform an extensive analysis of this task with different subset and overlap ratios, varying model and input sizes, and much more, establishing that the combination of NAU/NMU outperforms the NALU. We focus on the comparison of NPU, NMU, and NALU on the default parameters of Madsen and Johansen [2020] which sets the subset ratio to 0.5 and the overlap ratio to 0.25 (details in Tab. A4). We also compare to the NaiveNPU (without the relevance gate) to show how important the gating mechanism is for both sparsity and overall performance.

Fig. 7 plots training and testing errors over the number of non-zero parameters for all models and tasks. The addition column shows that all models except NaiveNPU learn $(+)$ on the training set, with the NMU converging to the sparsest models. The NMU also performs best on the testing set, with the NPU as a close second.
On $(\times)$, the best NMU models also outperform the NPU, but some of them do not converge at all, which is why the NPU has a better testing MSE on average (see also Tab. 1). The testing MSE of the NALU is so large that is excluded from the plot.
On $(\div, \sqrt{\cdot})$ the NPU clearly outperforms all other layers in MSE and sparsity. Generally, the difference between the NaiveNPU and the NPU is very significant and demonstrates how important the relevance gate is both for convergence and sparsity. The NPU effectively converts irrelevant inputs to 1s, while the NaiveNPU is stuck on the zero gradient plateau.

5 Conclusion

We introduced the *Neural Arithmetic Unit* that addresses the deficiencies of current arithmetic units: it can learn arbitrary power functions for positive, negative, and small numbers. We showed that the NPU outperforms its main competitor (NALU) and reaches performance that is on par with the multiplication specialist NMU. For tasks that are known to contain only addition and multiplication a stack of NAU and NMU can be a better choice than the NPU, because they might be easier to train. For more complicated problems that involve division or power functions, the NPU is the better choice.

Additionally we have demonstrated how the NPU can be used as an interpretable model that can recover governing equations of dynamical systems purely from data.

6 Statement of Broader Impact

Current neural network architectures can be problematic because they are black box models that are difficult to interpret. This becomes highly problematic if ML models are involved in high stakes decisions in e.g. criminal justice, healthcare, or control systems that affect lives. With the NPU we hope to contribute to the broad topic of interpretable machine learning, with a focus on scientific applications.

Additionally, learning to abstract (mathematical) ideas and extrapolate is a fundamental goal that might contribute to more reliable machine learning systems.

7 Acknowledgements

We thank the authors of the Julia packages Flux.jl [Innes et al., 2018] and DifferentialEquations.jl [Rackauckas and Nie, 2017]).

References

Kathleen Champion, Bethany Lusch, J. Nathan Kutz, and Steven L. Brunton. Data-driven discovery of coordinates and governing equations. *Proceedings of the National Academy of Sciences*, 116 (45):22445–22451, November 2019. ISSN 0027-8424, 1091-6490. doi: 10.1073/pnas.1906995116. URL `http://www.pnas.org/lookup/doi/10.1073/pnas.1906995116`.

Kaiyu Chen, Yihan Dong, Xipeng Qiu, and Zitian Chen. Neural Arithmetic Expression Calculator. *arXiv:1809.08590 [cs]*, September 2018. URL `http://arxiv.org/abs/1809.08590`. arXiv: 1809.08590.

Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. Neural Ordinary Differential Equations. *arXiv:1806.07366 [cs, stat]*, December 2019. URL `http://arxiv.org/abs/1806.07366`. arXiv: 1806.07366.

Stanislas Dehaene. *The Number Sense: How the Mind Creates Mathematics, Revised and Updated Edition*. Oxford University Press, April 2011. ISBN 978-0-19-991039-7. Google-Books-ID: 1p6XWYuwpjUC.

Lukas Faber and Roger Wattenhofer. Neural Status Registers. *arXiv:2004.07085 [cs, stat]*, April 2020. URL `http://arxiv.org/abs/2004.07085`. arXiv: 2004.07085.

C. R. Gallistel. Finding numbers in the brain. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 373(1740):20170119, February 2018. ISSN 0962-8436, 1471-2970. doi: 10.1098/rstb.2017.0119. URL `https://royalsocietypublishing.org/doi/10.1098/rstb.2017.0119`.

Alex Graves, Greg Wayne, and Ivo Danihelka. Neural Turing Machines. *arXiv:1410.5401 [cs]*, December 2014. URL `http://arxiv.org/abs/1410.5401`. arXiv: 1410.5401.

Michael Innes, Elliot Saba, Keno Fischer, Dhairya Gandhi, Marco Concetto Rudilosso, Neethu Mariya Joy, Tejan Karmali, Avik Pal, and Viral Shah. Fashionable Modelling with Flux. *arXiv:1811.01457 [cs]*, November 2018. URL `http://arxiv.org/abs/1811.01457`. arXiv: 1811.01457.

Łukasz Kaiser and Ilya Sutskever. Neural GPUs Learn Algorithms. *arXiv:1511.08228 [cs]*, March 2016. URL `http://arxiv.org/abs/1511.08228`. arXiv: 1511.08228.

Nal Kalchbrenner, Ivo Danihelka, and Alex Graves. Grid Long Short-Term Memory. *arXiv:1507.01526 [cs]*, January 2016. URL `http://arxiv.org/abs/1507.01526`. arXiv: 1507.01526.

Karol Kurach, Marcin Andrychowicz, and Ilya Sutskever. Neural Random-Access Machines. *arXiv:1511.06392 [cs]*, February 2016. URL `http://arxiv.org/abs/1511.06392`. arXiv: 1511.06392.

Brenden M. Lake and Marco Baroni. Generalization without systematicity: On the compositional skills of sequence-to-sequence recurrent networks. *arXiv:1711.00350 [cs]*, June 2018. URL `http://arxiv.org/abs/1711.00350`. arXiv: 1711.00350.

Guillaume Lample and François Charton. Deep Learning for Symbolic Mathematics. *arXiv:1912.01412 [cs]*, December 2019. URL `http://arxiv.org/abs/1912.01412`. arXiv: 1912.01412.

Andreas Madsen and Alexander Rosenberg Johansen. Measuring Arithmetic Extrapolation Performance. *arXiv:1910.01888 [cs]*, November 2019. URL `http://arxiv.org/abs/1910.01888`. arXiv: 1910.01888.

Andreas Madsen and Alexander Rosenberg Johansen. Neural Arithmetic Units. page 31, 2020.

Christopher Rackauckas and Qing Nie. DifferentialEquations.jl – A Performant and Feature-Rich Ecosystem for Solving Differential Equations in Julia. *Journal of Open Research Software*, 5:15, May 2017. ISSN 2049-9647. doi: 10.5334/jors.151. URL `http://openresearchsoftware.metajnl.com/articles/10.5334/jors.151/`.

Christopher Rackauckas, Yingbo Ma, Julius Martensen, Collin Warner, Kirill Zubov, Rohit Supekar, Dominic Skinner, and Ali Ramadhan. Universal Differential Equations for Scientific Machine Learning. *arXiv:2001.04385 [cs, math, q-bio, stat]*, January 2020. URL `http://arxiv.org/abs/2001.04385`. arXiv: 2001.04385.

Cynthia Rudin. Stop Explaining Black Box Machine Learning Models for High Stakes Decisions and Use Interpretable Models Instead. *arXiv:1811.10154 [cs, stat]*, September 2019. URL `http://arxiv.org/abs/1811.10154`. arXiv: 1811.10154.

David Saxton, Edward Grefenstette, Felix Hill, and Pushmeet Kohli. Analysing Mathematical Reasoning Abilities of Neural Models. *arXiv:1904.01557 [cs, stat]*, April 2019. URL `http://arxiv.org/abs/1904.01557`. arXiv: 1904.01557.

Daniel Schlör, Markus Ring, and Andreas Hotho. iNALU: Improved Neural Arithmetic Logic Unit. *arXiv:2003.07629 [cs]*, March 2020. URL `http://arxiv.org/abs/2003.07629`. arXiv: 2003.07629.

Mirac Suzgun, Yonatan Belinkov, and Stuart M Shieber. On Evaluating the Generalization of LSTM Models in Formal Languages. page 10, 2018.

Amirhossein Taghvaei, Tryphon T. Georgiou, Larry Norton, and Allen R Tannenbaum. Fractional SIR Epidemiological Models. preprint, Epidemiology, April 2020. URL `http://medrxiv.org/lookup/doi/10.1101/2020.04.28.20083865`.

Andrew Trask, Felix Hill, Scott Reed, Jack Rae, Chris Dyer, and Phil Blunsom. Neural Arithmetic Logic Units. *arXiv:1808.00508 [cs]*, August 2018. URL `http://arxiv.org/abs/1808.00508`. arXiv: 1808.00508.

Appendix

Table A1: Validation error of the different models (i.e. mean of each heatmap in Fig. 6). Each value is obtained by averaging the error of 20 models.

Task	NPU	NMU	NALU	iNALU	Dense
$+$	$\mathbf{0.2 \pm 0.11}$	$\mathbf{0.2 \pm 0.18}$	2.69 ± 0.22	2.18 ± 0.13	2.103 ± 0.04
$\times$	0.37 ± 0.23	$\mathbf{0.005 \pm 0.004}$	4.55 ± 0.2	3.453 ± 0.065	3.546 ± 0.035
$\div$	$\mathbf{0.23 \pm 0.13}$	11.399 ± 0.035	3.33 ± 0.18	2.54 ± 0.26	14.16 ± 0.23
$\sqrt{\cdot}$	$\mathbf{0.031 \pm 0.025}$	0.16 ± 0.002	0.034 ± 0.006	0.049 ± 0.011	0.084 ± 0.007

Table A2: Model definitions for the simple arithmetic task

Model	Layer 1	Layer 2	Layer 3
NPU	NAU$(2, 6)$	NPU$(6, 2)$	–
NMU	NAU$(2, 6)$	NMU$(6, 2)$	–
NALU	NALU$(2, 6)$	NALU$(6, 2)$	–
iNALU	iNALU$(2, 6)$	iNALU$(6, 2)$	–
Dense	Dense$(2, 10, \sigma)$	Dense$(10, 10, \sigma)$	Dense$(10, 2)$

Table A3: Model definitions for the large scale arithmetic task

Model	Layer 1	Layer 2
NPU	NAU$(100, 100)$	NPU$(100, 1)$
NPU	NAU$(100, 100)$	NPU$(100, 1)$
NMU	NAU$(100, 100)$	NMU$(100, 1)$
NALU	NALU$(100, 100)$	NALU$(100, 1)$

Table A4: Dataset parameters for the large scale arithmetic task.

Task	Input size	Subset ratio	Overlap ratio	Training range	Validation range
Add	100	0.5	0.25	Sobol(-1,1)	Sobol(-4,4)
Mult	100	0.5	0.25	Sobol(-1,1)	Sobol(-4,4)
Div	100	0.5	–	Sobol(0,0.5)	Sobol(-0.5,0.5)
Sqrt	100	0.5	–	Sobol(0,2)	Sobol(0,4)

Table A5: fSIR model definitions

Model	Layer 1	Layer 2	Layer 3
NPU	NPU$(3,\ h)$	NAU$(h,\ 3)$	–
NPU	NPU$_{\text{real}}(3,\ h)$	NAU$(h,\ 3)$	–
Dense	Dense$(2,\ h, \sigma)$	Dense$(h,\ h,\ \sigma)$	Dense$(h,\ 3)$