Neural Power Unit – A Bayesian Approach To Neural Arithmetic

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

Abstract

Common Neural Networks can approximate simple arithmetic operations, but fail to generalize beyond the range of numbers that were seen during training. A new class of units called *Neural Arithmetic Units* aim to overcome this difficulty, but are limited either to operate on positive numbers (NALU by Trask et al. [2018]), or can only represent simple addition and multiplication (NAU & NMU by Madsen and Johansen [2020]). We introduce a Neural Arithmetic Unit that operates on the full domain of real numbers and is capable of learning arbitrary power functions called *Neural Power Unit* (NPU). The NPU enables not only the learning simple arithmetic operations, but is also capable of learning e.g. Taylor expansions with very few parameters, which makes the NPU highly explainable. [1].

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]. This can be taken as a hint that arithmetic is an important ingredient to a successful, intelligent agent. State of the art neural networks are capable of learning simple 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.

The *Neural Arithmetic Logic Unit* (NALU) can learn addition, subtraction, multiplication, and division with two severe limitations: the inputs that the NALU is trained with have to be positive, and should be significantly larger than zero, otherwise the layer will not converge to a correct solution.

Madsen and Johansen [2020] introduce two new arithmetic layers, the *Neural Addition Unit* (NAU) and the *Neural Multiplication Unit* (NMU). Stacked, they can learn addition, subtraction, and multiplication and they converge for any input range. However, the NMU cannot learn division.

Including Neural Arithmetic Units in common neural networks, apart from the potential increase in extrapolation capabilities, has additional advantages. They promise to reduce the amount of parameters that are needed for a given task, can drastically improve the explainability of our models (more on the explainability of the NPU in Sec. 3). Additionally, the field of Scientific Machine Learning, which, in part, is concerned with learning differential equations [Rackauckas et al., 2020] could benefit from the inductive bias neural arithmetic.

[1] Implementation of neural arithmetic units: `https://github.com/nmheim/NeuralArithmetic.jl`

Preprint. Under review.

Our Contribution

We introduce a new arithmetic layer called the *Neural Power Unit* (NPU) which is capable of learning addition, subtraction, and arbitrary power functions, which includes multiplication and division. Our approach is inspired by the NALU, but overcomes its limitations to non-small positive numbers by leveraging the complex plane.

2 Related Work

2.1 Neural Arithmetic Logic Unit

Trask et al. [2018] have demonstrated how severe the extrapolation problem is even for the simplest arithmetic operations, such as summing or multiplying two numbers. In order to increase the power of abstraction of NNs they propose a *Neural Arithmetic Logic Unit* (NALU) which is capable of learning arithmetic addition, subtraction, multiplication, and division $\{+, -, \times, \div\}$ with stunning extrapolation accuracy. However, the NALU comes with the severe limitation not being able to handle negative inputs due to the logarithm in the multiplication part of the NALU:

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

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

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

Additionally the NALU has severe convergence problems for small inputs due to a non-smooth loss surface close to zero, as shown by Madsen and Johansen [2020].

should this be here already?

With the simple exemplary task of learning a function $f : \mathbb{R}^2 \to \mathbb{R}^2$

$$f(\boldsymbol{x}) = f(x, y) = [xy, x/y] = \boldsymbol{t} \tag{4}$$

we can demonstrate the strengths and weaknesses of the NALU. The NALU only learns successfully if it is trained on positive numbers, so we train with samples from $\boldsymbol{x} \in \mathcal{U}^2(0.1, 2)$ and plot the error $\epsilon = |\boldsymbol{t} - \hat{\boldsymbol{t}}|^2$ in Fig. 1.

2.2 Neural Multiplication Unit

A multiplication layer that can handle small and negative inputs was introduced by Madsen and Johansen [2020]. The *Neural Multiplication Unit* (NMU) is defined by Eq. 5 and is typically used in conjunction with the so-called *Neural Addition Unit* (NAU) in Eq. 6.

$$\text{NMU: } y_j = \prod_i M_{ij} z_i + 1 - M_{ij} \qquad M_{ij} = \min(\max(M_{ij}, 0), 1) \tag{5}$$

$$\text{NAU: } \boldsymbol{y} = \boldsymbol{A}\boldsymbol{x} \qquad A_{ij} = \min(\max(A_{ij}, 0), 1) \tag{6}$$

In both NMU and NAU the weights are clipped to $[0, 1]$, and typically regularized with $\mathcal{R}$:

$$\mathcal{R} = \sum_{ij} \min(W_{ij}, 1 - W_{ij}) \tag{7}$$

The combination of NAU and NMU can thus learn $\{+, -, \times\}$, but no division. This is shown in Fig. 1 as well. The NMU can perfectly represent the multiplication of the two inputs, but fails at learning division.

put this in acknowledgements all the results in this paper were create with the help of the following Julia packages: Rackauckas and Nie [2017] + (Flux.jl)

add in depth

3 Neural Power Units

Eq. 2 enables the NALU to learn multiplication. If the weight matrix $\boldsymbol{W}$ would not be restricted to values in the range $(-1, 1)$ it could in principle learn arbitrary power functions, because

$$z = \exp(w \log x) = x^w \text{ for } x > 0. \tag{8}$$

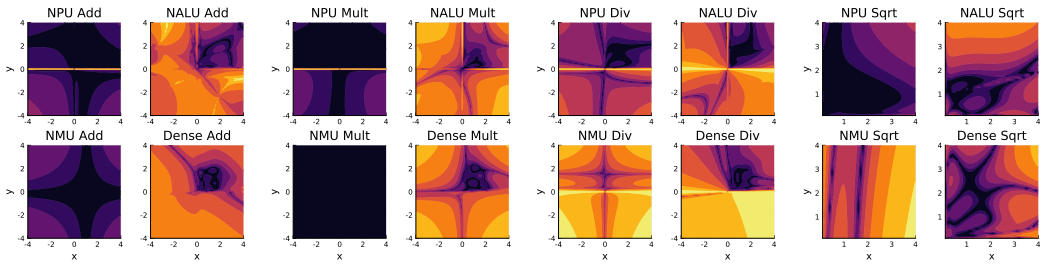

Figure 1: Comparison of different models learning the function $(x, y) \to (x + y, xy, x/y, \sqrt{(}x))$ with $(x, y) \in \mathcal{U}^2(0.1, 2)$. Each plot shows the mean-squared error on a logarithmic color scale ($\hat{\boldsymbol{t}} = \text{model}(\boldsymbol{x})$) of the best model of 20 runs. The Dense network learns all tasks, but fails to extrapolate. NALU is slightly better at extrapolating to positive numbers, but fails if one of the inputs is negative. The NMU successfully learns addition and multiplication, but not division and the square-root. Only the NPU learns all tasks and can extrapolate both to positive and negative numbers.

Table 1: Validation error of different models learning the function $(x, y) \to (x + y, xy, x/y, \sqrt{(}x))$ with $(x, y) \in \mathcal{U}^2(0.1, 2)$. The validation ranges are $(x, y) \in -4.1 : 0.2 : 4$ for addition, multiplication, and division, and $(x, y) \in 0.1 : 0.2 : 4$ for the square-root. Each value is the averaged absolute of 20 models.

Task	GatedNPU	NALU	NMU	Dense
Add	$\mathbf{0.18 \pm 0.11}$	2.69 ± 0.22	0.2 ± 0.18	2.103 ± 0.04
Mult	0.34 ± 0.23	4.55 ± 0.2	$\mathbf{0.005 \pm 0.004}$	3.546 ± 0.035
Div	$\mathbf{0.22 \pm 0.14}$	3.33 ± 0.18	11.399 ± 0.035	14.16 ± 0.23
Sqrt	$\mathbf{0.028 \pm 0.024}$	0.034 ± 0.006	0.16 ± 0.002	0.084 ± 0.007

However, this would complicate the convergence issues of the NALU even more. We therefore suggest to leverage the power of the complex plane, by using the complex logarithm and by promoting w to a complex number. This will enable us not only to lift the positivity constraint on x, but also provides and additional dimension that helps to avoid regions with an uninformative gradient signal. Introducing a complex weight matrix somewhere in a larger network would mean that the gradients in other layers would potentially become complex as well, which would result in converting the whole network to complex numbers. This seems like a high price to pay, but fortunately there is a way around this. We can output only the real part of the result $\text{Re}(z)$, but still maintain a complex w by treating it as a sum of real and imaginary part $w = w_r + iw_i$. We are thus interested in

$$\text{Re}(z) = \text{Re}(\exp(w \log x)), \tag{9}$$

with a real input x. The complex logarithm is defined as $\log z = \log r + i\theta$, which, for real inputs x, can be simplified to $\log z = \log r + ik\pi$, where $k \in \{0, 1\}$. With that Eq. 9 can be rewritten to

$$\begin{aligned} \text{Re}(z) &= \text{Re}(\exp((w_r + iw_i)(\log r + ik\pi))) && (10) \\ &= \exp(w_r \log r - w_i k\pi) \cos(w_i \log r + k\pi w_r), && (11) \end{aligned}$$

where we used Euler's formula[2].

rephrase this. do we have an educating example where it is obvious explicitly compute gradients?

3.1 Neural Power Unit (NPU)

The *Neural Power Unit* (NPU) with two weight matrices $\boldsymbol{W}_r$ and $\boldsymbol{W}_i$ then reads

$$\boldsymbol{r} = |\boldsymbol{x}|, \qquad k_i = \begin{cases} 0 & x_i \leq 0 \\ \pi & x_i > 0 \end{cases} \tag{12}$$

$$\boldsymbol{z} = \exp(\boldsymbol{W}_r \log \boldsymbol{r} - \boldsymbol{W}_i \boldsymbol{k}) \odot \cos(\boldsymbol{W}_i \log \boldsymbol{r} + \boldsymbol{W}_r \boldsymbol{k}) \tag{13}$$

fig.1 does not work as well with non-complex NPU, but I have not idea how to plot that

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

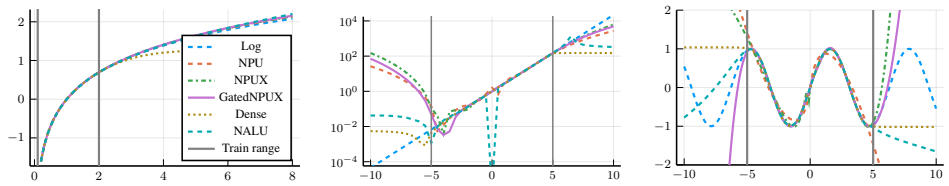

Figure 2: Function learning comparison of different layers.

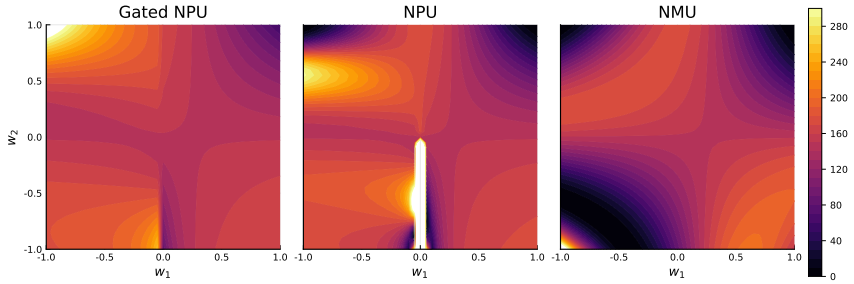

Figure 3: asdf

where $\boldsymbol{k}$ is a vector that is zero where $\boldsymbol{x}$ is positive and π where it is negative, $\boldsymbol{r}$ is the element-wise absolute value. In the final network we are only interested in real powers, so we recommend to strongly regularize the imaginary part of the weights $\boldsymbol{W}_i$.

3.2 Gated NPU

As pointed out by Madsen and Johansen [2020], the loss surface of the NALU, and thus also the NPU, is not always smooth. For some input and weight combinations it can happen that the loss close to zero becomes very large as shown in Fig. 3. However, most applications are dealing with batches of data which average out these cases, as shown in Fig. 4 for a batch size of 32.

In practice we encountered difficulties of obtaining sparse solutions with the NPU. If the inputs to an NPU unit are small, the gradient becomes practically zero, as we can see in Fig. 5

4 Experimetns

This can be demonstrated by training a traditional, dense NN to approximate three functions

$$f(x) = e^x, \qquad g(x) = \log(x), \qquad h(x) = \sin(x) \tag{14}$$

Neural Arithmetic Units aim to overcome this problem with units that are able to represent simple arithmetic operations exactly. By composition of these units it becomes possible to learn e.g. exponentials, logarithms, and power functions. This promises to improve on the extrapolation capabilities

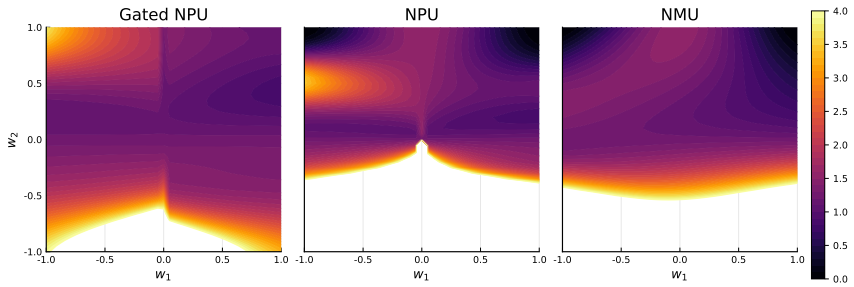

Figure 4: asdfadsf

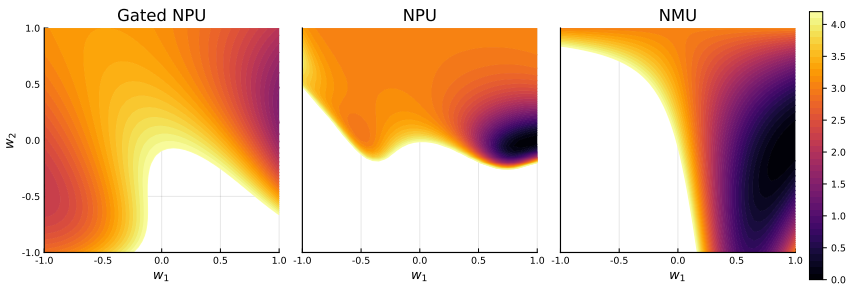

Figure 5: learning identity with small inputs

of NNs even on tasks that are beyond artificial arithmetic tasks such as learning multiplication, or a certain power function .

cite some examples

4.1 10 param func

4.2 Gradient surfaces

References

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.

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`.

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.

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.

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

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.