Extending Neural Networks Using Polynomial Arithmetic

Tomas Ukkonen

Novel Insight Research, Finland

Abstract

The multilayer neural network architecture has seen relatively little number-theoretic research. However, improving fundamental computing capacity by extending to the more extensive set of possible numbers and functions could improve machine learning results. Typically, networks often use real and sometimes complex numbers, and extending numbers to contain dimensional information may improve the abstract processing of real-world data. The resulting polynomial ring generalizes multiplication to convolution while the neural network is as close as possible the standard one. Experiment using synthetic problem compare performance between real-valued standard and polynomial ring implementations.

Keywords: neural networks, polynomial rings, number theory, cryptography

1. Introduction

Number theoretic research in neural networks research has not been very common although more complex computations and functions could improve optimization results. This happens without the need to do significant modifications to the standard architectures (fully connected, recurrent) or core algorithms (backpropagation, gradient descent, activation functions) [1]. In research, neural networks have been typically only extended to process complex numbers or sometimes quaternions [2, 3, 4, 5].

Email address: tomas.ukkonen@novelinsight.fi (Tomas Ukkonen)
URL: www.novelinsight.fi (Tomas Ukkonen)

2. Theoretical background

This paper's motivation to extend real numbers to a polynomial ring is an intuition that real numbers can be extended to contain dimensionality information using hyperreal numbers, non-standard analysis, and fractal geometry [6, 7, 8]. A quantity of dimension d can be the multiplication of a real-valued scalar a and a volume of d-dimensional hypercube r^d, where r is the arbitrary (unit) length of one side of the hypercube. One can then make a somewhat simple definition of "multidimensional" numbers s.

$$s = a_0 * r^0 + a_1 * r^1 + a_2 * r^2 ... \tag{1}$$

Compared to the magnitude of real numbers, r's length is infinite and outside of the set $\mathbb{R}$, so the components are in the same direction but perpendicular to each other in a vector space sense because there is no scalar $a \in \mathbb{R}$ to scale different components to be the same. It seem to be possible to extend this line of reasoning formally to cases where d is real-valued, meaning fractal dimensions, or abstractly to imaginary and negative dimensions (r^{-1} is infinitesimal dr) and, in addition to multiplication, define a stretching operation c to an object $a_{new} * r^d = a_{old} * (c * r)^d = a_{old} * c^d * r^d$. By incorporating dimensional information into a neural network, it might better abstractly process real-world data.

To have a well-defined closed number system that can compute using computers, I restrict dimensions to be natural numbers $\mathbb{N}$ and define a real polynomial ring $R[X]/(X^K - 1)$ [9] using the following definitions.

$$\begin{aligned} s &= \sum_{d=0}^{K-1} a_d * r^d, a_d \in \mathbb{R}, d \in \mathbb{N} \\ s_1 + s_2 &= \sum_{d=0}^{K-1} (a_{d_1} + a_{d_2}) * r^d \\ s_1 * s_2 &= \sum_{d_1=0, d_2=0}^{K-1, K-1} a_{d_1} * a_{d_2} * r^{d_1 + d_2 (mod) K} \end{aligned} \tag{2}$$

By choosing modulo operation K to be a prime number, it is easy to see that the polynomial ring becomes a field with a finite K number of components. Multiplication operation now leads to a circular convolution of the

coefficients. The circular convolution and its inverse, the division operation, can be efficiently calculated using discrete Fourier transform [10]. By making dimensions d circular, the comparability of the numbers is lost even when using real coefficients. However, if numbers/the circular convolution is properly zero-padded, it is often possible to calculate the standard convolution. The circular convolution operation processes each component symmetrically, ignoring dimensional information. This asymmetry is later (weakly) imposed by applying stretching operation when initializing the neural network's weights so that weights are approximately zero-padded.

3. Neural network architectures using polynomial ring numbers

3.1. Linear model

The simplicity of the linear optimization is that the function $\mathbf{y} = \mathbf{A}*\mathbf{x}+\mathbf{b}$ has only one easily solvable MSE optimum, and it is the global optimum of the problem. In practice, linear functions are often too simple for many practical problems, but if we can do a non-linear number theoretic extension to real numbers fulfilling field axioms, it is possible to solve the global optimum of non-linear problems. Unfortunately, for polynomial field numbers s and real-valued data, the non-real parts of the coefficients are always zero, meaning that it is impossible to extend calculations non-linearly.

In data analytics, however, it is common to discretize real-valued variables using one-hot encoding, which maps real numbers to higher dimensional vectors, after which the global optimum of an approximated problem is solvable. In this paper, other number theoretic possibilities are not studied. A multilayer neural network is used instead, in which non-linearities make it possible to process real-valued data using extended number systems.

3.2. Neural network model

To keep simple optimization methods like gradient descent functional, and the neural network was kept as close to linear as possible. The densely connected neural network's layers use a leaky ReLU non-linearity [11, 12] except at the last layer, which is entirely linear. The ReLU activation function is extended to polynomial fields by applying ReLU non-linearity $f(s)$ only to the first dimension and not changing the other components. For polynomial field numbers s, a heuristical derivate of ReLU was used, which gives much better optimization results than calculating derivate directly from the definition.

$$
\begin{aligned}
f(s)_j &= \begin{cases} max(as_j, s_j), & \text{if } j = 0\,, \\ s_j, & \text{if } j > 0\,. \end{cases} \\
df/ds &\approx \begin{cases} a, & \text{if } s_0 < 0\,, \\ 1, & \text{if } s_0 \geq 0\,. \end{cases}
\end{aligned}
\tag{3}
$$

Derivates are well-defined for linear algebra operations when using polynomial field numbers s. This is because derivates are not dependent on what direction zero is approached.

$$
df(s)/ds = \lim_{h \to 0}((a * (s + h) + b) - a * s)/h = lim_{h \to 0} a = a \tag{4}
$$

To calculate the derivate of mean squared error using polynomial field, MSE is written using an operator $\star$, which multiplies each component of the polynomial field.

$$
a \star b = \sum_{d=0}^{K-1} a_d b_d r^d \tag{5}
$$

We can then write MSE error as a $\star$ product of $\mathbf{1} = 1 * r^0$, which selects the real part of the error term, and a $\star$ product of error delta terms that squares error values. Equation 6 is for simplicity written using dimensions j but can be generalized to vectors and its derivate calculated from Jacobian matrix.

$$
MSE(\mathbf{w}) = \mathbf{1} \star (\frac{1}{2N} \sum_j \sum_{i=0}^{N-1} (f_j(\mathbf{x_i}|\mathbf{w}) - y_{ij}) \star (f_j(\mathbf{x_i}|\mathbf{w}) - y_{ij})) \tag{6}
$$

Now, because of the close resemblance of our polynomial ring to the Fourier transform, it is possible to transform $\star$ products to convolutions by calculating the Fourier transform $\mathfrak{F}$ of the terms. These convolutions and Fourier transforms can be easily derived because they are not dependent on the neural network's weight parameters $\mathbf{w}$. In gradient equations 7, a $\circ$ operator marks circular convolution operation.

$$MSE(\mathbf{w}) = \mathbf{1} \star MSE'(\mathbf{w})$$
$$\frac{dMSE'(\mathbf{w})}{d\mathbf{w}} = \mathfrak{F}^{-1}(\frac{1}{N}\sum_{j}\sum_{i=0}^{N-1}\mathfrak{F}(f_j(\mathbf{x_i}|\mathbf{w}) - y_{ij}) \circ \mathfrak{F}(\frac{df_j(\mathbf{x_i}|\mathbf{w})}{d\mathbf{w}})) \quad (7)$$
$$\frac{dMSE(\mathbf{w})}{d\mathbf{w}} = \mathfrak{F}^{-1}(\mathfrak{F}(\mathbf{1}) \circ \mathfrak{F}(\frac{dMSE'(\mathbf{w}))}{d\mathbf{w}})$$

In experiments, I always set non-real parts of the error vector to be zero. The idea that stretching an object should not change the properties of an object is used to apply stretching operation $c = 0.25$ to initial random weights. This means that larger dimensional weights are initially close to zero, and process data in low dimensions. Therefore computations start with zero padding of the convolution operations between numbers but allow the network to use the circular convolution and higher dimensions if it reduces MSE error.

3.3. Gradient descent algorithm

Algorithm 1 describes a modified gradient descent algorithm that trains polynomial ring neural networks. It uses costly adaptive step length and sometimes enters worse solutions. The mechanism allows the algorithm to escape from local minimums into what polynomial ring minimization often gets stuck. The algorithm rarely converges but tries worse solutions looking for a way away from the local minimum. Therefore execution of the algorithm was stopped after 500-2000 iterations when there were no more improvements or when execution time was 2.5-48 hours.

4. Experimental results

4.1. Experiment 1

The first experiment uses a synthesized problem and a small number of parameters. The amount of synthetic data is only $N = 1000$, so even our unoptimized algorithm would run the experiment in 24 hours (and similarly also in Experiment 2). The training did not do early stopping and overfitted to data. The standard neural network implementation is TensorFlow with Adam optimizer which is comparated against our gradient descent algorithm. The non-linearities of the first layers of the neural network are ReLU activation functions, and the last layer is linear. Here it is interesting how a neural

Algorithm 1 Gradient Descent algorithm for polynomial ring neural network

```
1: procedure GradientDescent
2:     e ← ∞
3:     while e > error_limit do
4:         e ← calculate_error(w)
5:         g ← calculate_gradient(w)
6:         errors ← ∞
7:         epsilon ← 0.02
8:         iters ← 0
9:         while last(errors) > e & iters < 500 do
10:            epsilon ← epsilon/2
11:            w_next ← w + epsilon * g
12:            iters ← iters + 1
13:            errors ← calculate_error(w_next)
14:            r ← (random()%40)
15:            if last(errors) < 2 * average(errors, 20) & r == 0 then
16:                errors ← 0 ; goes to worse solution with a 2.5% probability
17:        w ← w_next
```

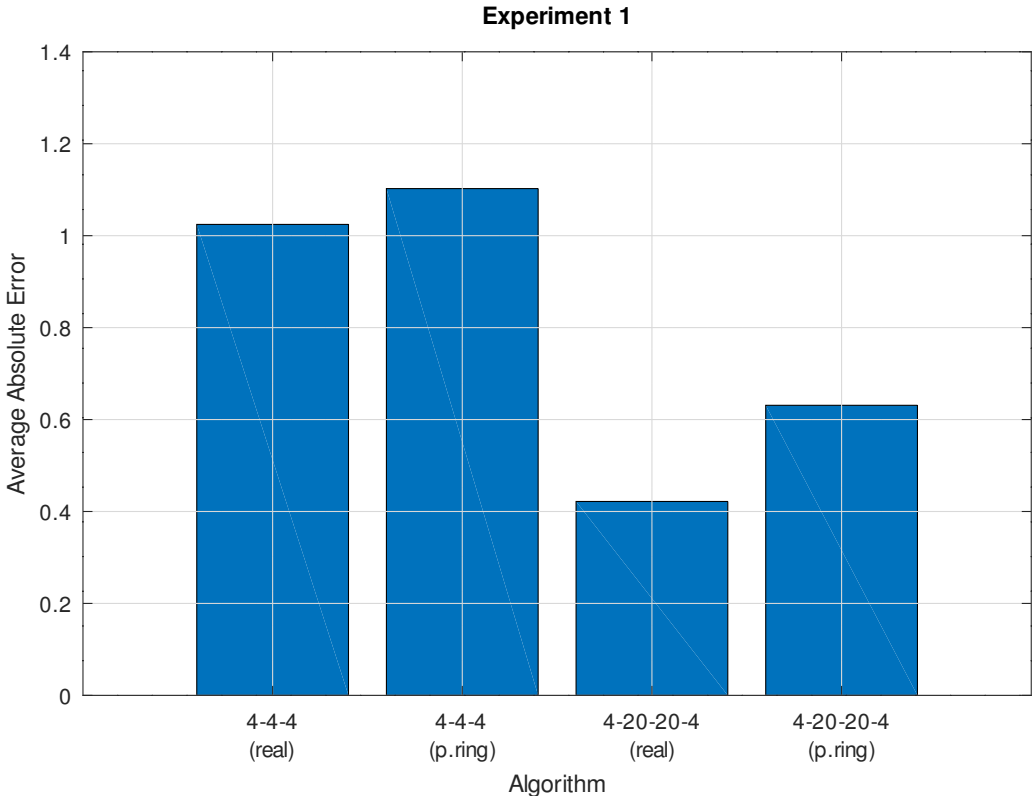
network learns simple non-linear tasks. The input variables $\mathbf{x} \sim N(\mathbf{0}, 4^2\mathbf{I})$ are mapped to values $\mathbf{y}$ using the following equations.

$$\begin{aligned} y_1(\mathbf{x}) &= \sin(f * x_1 * x_2 * x_3 * x_4) \\ y_2(\mathbf{x}) &= sign(x_4) * a^{x_1/(|x_3|+1)} \\ y_3(\mathbf{x}) &= sign(\cos(w * x_1)) + sign(x_2) + sign(x_4) \\ y_4(\mathbf{x}) &= x_2/(|x_1|+1) + x_3 * \sqrt{|x_4|} + |x_4 - x_1| \\ f &= 10.0, w = 10.0, a = 1.1 \end{aligned} \tag{8}$$

Results of overfitting optimization are in Table 1. The algorithm computed results five times, and the best value was chosen. The problem was too complicated for a small two-layer $4-4-4$ neural network, and algorithms couldn't learn the dataset. However, even in this case, standard neural network implementation performed better for normal data than polynomial ring neural networks. The difference remained even with larger neural networks while the computing time to converge to a minimum error was much bigger, 5 minutes with a standard TensorFlow implementation and 12-24 hours with the polynomial ring neural network. This result shows that polynomial ring neural networks still require more work to optimize the non-linearity, the number of dimensions, initialization of weights, and the learning algorithm if they would be usable as general problem solvers in machine learning.

Architecture	Error
4-4-4 (real-valued)	1.0245
4-4-4 (11 dimensional numbers)	1.10242
4-20-20-4 (real-valued)	0.4218
4-20-20-4 (11 dimensional numbers)	0.6310

Table 1: Minimum absolute error found with different neural network architectures.



5. Discussion

In this paper, I have described interesting number theoretic extension to neural networks. The results imply that the extended neural network is able to learn much more complex functions with smaller error but is plagued by problems such as getting stuck to local optimums which require use of modified heuristical derivate in non-linearity and modified gradient descent algorithm. More research is needed to solve these problems.

Although it has not been studied in this paper, the convolutional nature of the number system could make it helpful in the real-world processing of audio and picture signals. Another line of further study would be to try the effect of different nonlinearities to the maximum possible network depth. In our experiments, the depth was only a few layers, but the current framework scales up to 40 layers using a residual neural network.

6. Funding and further information

This research has not received any funding or grants from any sources. Polynomial ring neural network implementation will be later published as open source as part of Dinrhiw2 C++ machine learning software library [13].

References

[1] S. Haykin, Neural Networks: A Comprehensive Foundation, 2nd Edition, Prentice-Hall International, 1999.

[2] T. Nitta, A back-propagation algorithm for complex numbered neural networks, in: Proceedings of 1993 International Conference on Neural Networks, IEEE, 1993. doi:10.1109/IJCNN.1993.716968.

[3] K. Tachibana, K. Otsuka, Wind prediction performance of complex neural network with relu activation function, in: 2018 57th Annual Conference of the Society of Instrument and Control Engineers of Japan, IEEE, 2018. doi:10.23919/SICE.2018.8492660.

[4] A. Hirose, Complex-Valued Neural Networks: Theories and Applications, World Scientific Publishing Cp. Ptd. Ltd., 2003.

[5] C. J. Gaudet, A. S. Maida, Deep quaternion networks, in: 2018 International Joint Conference on Neural Networks, IEEE, 2018. doi:10.1109/IJCNN.2018.8489651.

[6] H. J. Keisler, The hyperreal line, in: Real Numbers, Generalizations, of the Reals, and Theories of Continua, Vol. 242, Springer, 1994. doi:10.1007/978-94-015-8248-3_8.

[7] A. Robinson, Non-Standard Analysis, revised edition, Princeton University Press, 1996.

[8] K. Falconer, Fractal Geometry: Mathematical Foundations and Applications, 2nd Edition, John Wiley & Sons Ltd., 2003.

[9] K. B. Moore, Discrete Mathematics Research Progress, Nova Science Publishers Inc., 2008.

[10] S. K. Mitra, Digital Signal Processing: A Computer-Based Approach, McGraw-Hill series in electrical and computer engineering, 1998.

[11] R. Hahnloser, R. Sarpeshkar, M. Mahowald, R. Douglas, H. S. Seung, Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit, Nature 405 (6789) (2000) 947–951.

[12] X. Glorot, A. Bordes, Y. Bengio, Deep sparse rectifier neural networks, in: Proceedings of the 14th International Conference on Artificial Intelligence and Statistics (AISTATS), 2011.

[13] T. Ukkonen, Dinrhiw2: C++ machine learning library (2022).
URL `https://github.com/cslr/dinrhiw2`