Extending neural networks to use dimensional numbers

Tomas Ukkonen

Novel Insight, Finland

Abstract

The multilayer neural network architecture has seen relatively little number theoretic research despite the fact that improving fundamental computing capacity by extending to the larger set of possible numbers and functions could improve machine learning results. Currently, typical networks often use real and sometimes complex numbers. Extending numbers to contain dimensional information may improve abstract processing of real world data. The resulting polynomial field generalizes multiplication to convolution while neural network is modified to learn to be invariant to topological operation of stretching. Experiments using synthetic and text-based data sets compare performance to real and complex value implementations.

Keywords: neural networks, polynomial fields, number theory, natural language processing

1. Introduction

Number theoretic work in neural networks research has not been very common although more complex computations and functions could improve optimization results without need to do major modifications to standard architectures (fully connected, convolutional, recurrent) or core algorithms (backpropagation, stochastic gradient descent, activation functions) [1]. In research, neural networks has 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 or complex numbers to a polynomial ring is an intution that real numbers can be extended to contain dimensionality information by using hyperreal numbers, non-standard analysis and fractal geometry [6, 7, 8]. A quantity of dimension d can be written to be multiplication of a real valued scalar a and a volume of d-dimensional hypercube r^d where r is arbitrary (unit) length of one side of the hypercube. One can then make somewhat naive definition of "multidimensional" numbers s.

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

In comparison to the length of real line 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 are no scalar $a \in \mathbb{R}$ to scale different components to be same. It is possible to formally extend this line of reasoning 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 strectching operation c to an object $a_{new} * r^d = a_{old} * (c * r)^d = a_{old} * c^d * r^d$. By incorporating dimensional information to a neural network it is hoped to perform better in abstract processing of real world data.

To have a well-defined closed number system which can be computed using computers we restrict dimensions to be natural numbers $\mathbb{N}$ and define a complex polynomial field $R[X]/(X^K - 1)$ [9] using the following definitions

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

By choosing modulo operation K to be a prime number it is easy to see that the polynomial ring becomes a field with finite K number of components. Multiplication operation leads now 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 we lose the comparability of the numbers even when using real coefficients but if numbers/the circular convolution is properly zero padded it is often possible to calculate the normal convolution. The convolution operation processes each component symmetrically ignoring dimensional information. This symmetry is later broken by applying stretching operation to data processed by a neural network.

3. Neural network architectures using polynomial field 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 global optimum of non-linear problem. Unfortunately, for polynomial field numbers s and real valued data the non-real parts of the coefficients are always zero meaning that it is not possible 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 can be solved somewhat similar to support vector machines **addreference**. In this paper, further number theoretic possiblities were not studied and a multilayer neural network was used instead in which non-linearities make it possible to process real valued data using extended number systems.

3.2. Deep recurrent neural network model

To keep simple optimization methods like stochastic gradient descent working and to combat against vanishing gradient problem the neural network was kept as closely to linear as possible. The densely connected neural network's layers uses a leaky ReLU non-linearity [11, 12] except at the last layer which was chosen to be fully linear. The ReLU activation function was extended to complex numbers and polynomial fields by applying ReLU non-linearity component-wise. The simple ReLU activation function makes it possible to calculate easily derivates and the Jacobian matrix using backpropagation. For complex numbers z and especially for polynomial field numbers s we assumed that the component-wise ReLU is close to a linear function which goes through origo meaning its slope or the first derivate can be aproximated easily

$$f(z) = a * z, df/dz \approx a(z) = f(z)/z = max(0.01 * z, z)/z \tag{3}$$

Note that derivates are well-defined for linear algebra when using polynomial field numbers s because derivates are not dependend from what direction zero is approached.

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

For calculating the derivate of mean squared error using complex polynomial field numbers MSE is generalized to be real valued function ($\sigma_i(s)$ selects i:th complex component of s and its generalization is to all compomnents is identity function $\sigma(s) = s$). Because variables in the inner product are complex Wirtinger-calculus was used to calculate derivates [13]. In the gradient we calculate component-wise multiplication of s (marked with combined transpose/vector product $\mathbf{x}^{T*}$) and use the fact that we can then combine $\sigma_i(s)$ operations to a single $\sigma(s)$ operation.

$$\begin{gathered}
MSE(\mathbf{w}) = E_{\mathbf{xy}}\{\frac{1}{2}\sum_{i=0}^{K-1} ||\sigma_i(\mathbf{f}(\mathbf{x}|\mathbf{w})) - \sigma_i(\mathbf{y})||^2\} \\
\frac{dMSE(\mathbf{w})}{d\mathbf{w}} = E_{\mathbf{xy}}\{\sum_{i=0}^{K-1} \sigma_i(\mathbf{f}(\mathbf{x}|\mathbf{w}) - \mathbf{y})^T \overline{\frac{d\sigma_i(\mathbf{f}(\mathbf{x}|\mathbf{w}))}{d\mathbf{w}}}\} \\
\frac{dMSE(\mathbf{w})}{d\mathbf{w}} = E_{\mathbf{xy}}\{\sigma(\mathbf{f}(\mathbf{x}|\mathbf{w}) - \mathbf{y})^{T*} \overline{\frac{d\sigma(\mathbf{f}(\mathbf{x}|\mathbf{w}))}{d\mathbf{w}}}\} \\
\frac{dMSE(\mathbf{w})}{d\mathbf{w}} = E_{\mathbf{xy}}\{(\mathbf{f}(\mathbf{x}|\mathbf{w}) - \mathbf{y})^{T*} \overline{\frac{d\mathbf{f}(\mathbf{x}|\mathbf{w})}{d\mathbf{w}}}\}
\end{gathered} \tag{5}$$

In experiments we always set non-real parts of the error vector to be zero meaning that we only optimize for the correct real part of the output. In our second experiment (Section 4.2) we used a simple recurrent neural network for predicting and generating text sequences. To keep experiments simple a normal feedforward network with a single feedback loop from output to input was used instead of LSTMs. Jacobian matrix was computed by applying BPTT algorithm to recurrent neural networks.

To improve generalization and to simulate practical optimization tasks the dropout heuristic with 20% dropout rate of hidden layer neurons was used [14]. When using polynomial field numbers the idea that stretching an object should not essentially change properties of an object is used to apply random scretching operation $c \in [0.2, 1.4]$ as a generalized dropout operation to hidden layers. This operation forces network to use dimensionality information and because high dimensional components change the fastest it causes network to process data in low dimensions.

4. Experimental results

4.1. Experiment 1

In the first experiment simple synthetic data and small number of parameters was used. The amount of synthetic data was chosen to be large $N = 10^6$ so a raw optimization performance without early stopping or dividing data into training and testing sets was measured. The neural network is a two-layer $4-4-4$ dense neural network where the first layer have ReLU activation function and the last layer is linear. Here we are interested how much better neural network learns simple non-linear task. The input variables $\mathbf{x} \sim N(\mathbf{0}, 4^2\mathbf{I})$ are mapped to values $\mathbf{y}$ using equations

$$\begin{gathered} y_1(\mathbf{x}) = \sin(f * x_1 * x_2 * x_3 * x_4) \\ y_2(\mathbf{x}) = sign(x_4) * a^{x_1/x_3} \\ y_3(\mathbf{x}) = sign(\cos(w * x_1)) + sign(x_2) + sign(x_4) \\ y_4(\mathbf{x}) = x_2/x_1 + x_3 * \sqrt{x_4} + |x_4 - x_1| \end{gathered} \tag{6}$$

4.2. Experiment 2

The second experiment uses a recurrent neural network (Section 3.2) to predict text sequences. We use dataset X. Add references to OpenAI/Google Brain GPT-3 and GPT-2 papers and earlier papers using recurrent neural networks in academy.

5. Discussion

In this paper we have described number theoretic extension to neural networks which can be used to create neural network with arbitrary large number of parameters using same number of variables and layers. Although it has not been studied in this paper, the convolutional nature of the number system could make it useful for processing real world signals like audio or pictures.

6. Funding

This research has not received any funding or grants from any sources.

References

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

[2] T. Nitta, A back-propagation algorithm for complex numbered neural networks, in: Proceedings of 1993 International Conference on Neural Networks, IEEE, 1993 (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 (2018). doi:10.23919/SICE.2018.8492660.

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

[5] C. J. Gaudet, A. S. Maida, Deep quaternion networks, in: 2018 International Joint Conference on Neural Networks, IEEE, 2018 (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 (1994). doi:10.1007/978-94-015-8248-3_8.

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

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

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

[10] S. K. Mitra, Digital Signal Processing: A Computer-Based Approach, McGraw-Hill series in electrical and computer engineering, 1998 (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 (2000).

[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 (2011).

[13] K. Kreutz-Delgado, The complex gradient operator and the cr-calculus, arXiv.org preprint 0906.4835v1 (2009).

[14] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov, Dropout: a simple way to prevent neural networks from overfitting, The Journal of Machine Learning Research 15 (1) (2014) 1929–1958 (2014).