AMOEBA - A SELF-ORGANISING NEURAL NETWORK

DR. LI, NGO MAN DESMOND

Abstract. In this article, we propose a new artificial neural network architecture by growing

Keywords. AI, Artificial Neural Networks, Machine Learning, LLM

Eleven AI. AI
Website: https://
Email: desmondli@cryptomattock.com
Document ID: P1903251

Contents

1. Introduction

At the time of writing, Artificial Intelligence (AI) is one of the hottest topic of discussion. The advent of Large Language Models (LLM) like ChatGPT or DeepSeek [4, 2] only sparked more public interest and institutional investments into AI. These product also lead to new problem solving methods, predictive models, and new business opportunities. But in the heart of neural networks, there lies an arbitrariness on the depth and number of layers required. Before we continue our discussion, we should first ask the question, "why an artificial neural network?". The answer to this question is because of a theorem in functional analysis, namely the Universal Approximation Theorem Phase 1: a) Study the AI model for a 1D problem b) Develop an algorithm for neuronetwork expansion c) Run tests to prove the theory works If this works, this can be one academic paper.

2. Architecture

In this section, we talk about the architecture design of our neural network. To express our neural network properly, we make use of the *Hadamard product*. The Hadamard product between two matrices is just the element-wise product, the following definition will make this precise.

Definition 2.1. Let R be a ring, $m, n \in \mathbb{N}$, and $R^{m\times n}$ be the set of matrices with elements in the ring R. For any two matrices $A, B \in R^{m\times n}$, the Hadamard product between A and B is defined by,

$$(A \odot B)_{ij} := a_{ij} b_{ij}.$$

We would also like to extend the definition to matrix functions.

Let $A \in R^{m\times n}$ and $\boldsymbol{\sigma} : R^{m\times n} \to R^{m\times n}$, we define the Hadamard product between A and $\boldsymbol{\sigma}$ by,

$$(A \odot \boldsymbol{\sigma})_{ij} := a_{ij}\sigma_{ij}$$

and

$$\boldsymbol{\sigma} \odot A := \boldsymbol{\sigma}(A)$$

where the latter case we will drop the $\odot$ whenever the context is clear. Note that under this definition, the Hadamard product between a matrix and a matrix function is not commutative.

With the Hadamard product defined, we may now introduce our neural network architecture. Given a function $\mathbf{f} : R^n \to R^m$, our goal is to approximate this function systematically but by only adjusting a finite set of values. Our model follows the usual feedforward structure, but allowing the activation function to be tunable. The idea to have a learnable activation is not new, our model is build upon work done by Agostinelli, Manessi, et al. in [1, 3].

Definition 2.2. An *Amoeba Neural Network (ANN)* over a ring R is the pentuple $(\mathcal{I}, \{A_j\}_{j\in\mathcal{I}}, \{\mathbf{b}_j\}_{j\in\mathcal{I}}, \{C_j\}_{j\in\mathcal{I}}, \boldsymbol{\sigma})$, where $\mathcal{I} = \{1, 2, 3, ..., m\}$ is an index set, $\{A_j\}_{j\in\mathcal{I}}$ and $\{C_j\}_{j\in\mathcal{I}}$ are sets of matrices called the *weights*, $\{b_j\}_{j\in\mathcal{I}}$ is a set of vectors called the *bias*, all of which are indexed by the index set $\mathcal{I}$. Finally, $\boldsymbol{\sigma} : R^n \to R^n$ is a vector function on R called the *activation function*. By default, we will assume that $\boldsymbol{\sigma}_1 = \mathrm{id}$.

For a given vector $\mathbf{x}_0$ in R called the *input vector*, we may define a sequence by,

$$\mathbf{x}_{k+1} = (C_{k+1}\boldsymbol{\sigma}) \odot (A_{k+1}\mathbf{x}_k + \mathbf{b}_{k+1})$$

and $\mathbf{x}_m$ is then called the *output vector*. As a result, the Amoeba Neural Network defines a vector function on the ring R.

References

[1] F. Agostinelli, M. Hoffman, P. Sadowski, and P. Baldi, *Learning activation functions to improve deep neural networks*, 2015.

[2] DeepSeek-AI, D. Guo, D. Yang, H. Zhang, J. Song, R. Zhang, R. Xu, Q. Zhu, S. Ma, P. Wang, X. Bi, X. Zhang, X. Yu, Y. Wu, Z. F. Wu, Z. Gou, Z. Shao, Z. Li, Z. Gao, A. Liu, B. Xue, B. Wang, B. Wu, B. Feng, C. Lu, C. Zhao, C. Deng, C. Zhang, C. Ruan, D. Dai, D. Chen, D. Ji, E. Li, F. Lin, F. Dai, F. Luo, G. Hao, G. Chen, G. Li, H. Zhang, H. Bao, H. Xu, H. Wang, H. Ding, H. Xin, H. Gao, H. Qu, H. Li, J. Guo, J. Li, J. Wang, J. Chen, J. Yuan, J. Qiu, J. Li, J. L. Cai, J. Ni, J. Liang, J. Chen, K. Dong, K. Hu, K. Gao, K. Guan, K. Huang, K. Yu, L. Wang, L. Zhang, L. Zhao, L. Wang, L. Zhang, L. Xu, L. Xia, M. Zhang, M. Zhang, M. Tang, M. Li, M. Wang, M. Li, N. Tian, P. Huang, P. Zhang, Q. Wang, Q. Chen, Q. Du, R. Ge, R. Zhang, R. Pan, R. Wang, R. J. Chen, R. L. Jin, R. Chen, S. Lu, S. Zhou, S. Chen, S. Ye, S. Wang, S. Yu, S. Zhou, S. Pan, S. S. Li, S. Zhou, S. Wu, S. Ye, T. Yun, T. Pei, T. Sun, T. Wang, W. Zeng, W. Zhao, W. Liu, W. Liang, W. Gao, W. Yu, W. Zhang, W. L. Xiao, W. An, X. Liu, X. Wang, X. Chen, X. Nie, X. Cheng, X. Liu, X. Xie, X. Liu, X. Yang, X. Li, X. Su, X. Lin, X. Q. Li, X. Jin, X. Shen, X. Chen, X. Sun, X. Wang, X. Song, X. Zhou, X. Wang, X. Shan, Y. K. Li, Y. Q. Wang, Y. X. Wei, Y. Zhang, Y. Xu, Y. Li, Y. Zhao, Y. Sun, Y. Wang, Y. Yu, Y. Zhang, Y. Shi, Y. Xiong, Y. He, Y. Piao, Y. Wang, Y. Tan, Y. Ma, Y. Liu, Y. Guo, Y. Ou, Y. Wang, Y. Gong, Y. Zou, Y. He, Y. Xiong, Y. Luo, Y. You, Y. Liu, Y. Zhou, Y. X. Zhu, Y. Xu, Y. Huang, Y. Li, Y. Zheng, Y. Zhu, Y. Ma, Y. Tang, Y. Zha, Y. Yan, Z. Z. Ren, Z. Ren, Z. Sha, Z. Fu, Z. Xu, Z. Xie, Z. Zhang, Z. Hao, Z. Ma, Z. Yan, Z. Wu, Z. Gu, Z. Zhu, Z. Liu, Z. Li, Z. Xie, Z. Song, Z. Pan, Z. Huang, Z. Xu,

Z. Zhang, and Z. Zhang, *Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning*, 2025.

[3] F. Manessi and A. Rozza, *Learning combinations of activation functions*, in 2018 24th International Conference on Pattern Recognition (ICPR), IEEE, Aug. 2018, p. 61–66.

[4] Z. Shao, D. Dai, D. Guo, B. L. B. Liu), Z. Wang, and H. Xin, *Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model*, ArXiv, abs/2405.04434 (2024).