Southeast Asian Journal of Sciences, Vol. 11, No 2 (2025), pp. 74-80

APPROXIMATING TRUST FUNCTIONS FROM MODELS IN NEURON NETWORK AND LIE ALGEBRA

Dinh Que Tran

Department of Information Technology
Posts and Telecommunications Institute of Technology (PTIT)
Hanoi, Vietnam
E-mail: quetd@ptit.edu.vn

Abstract

This paper is to describe a tensor representation of features needed for trust computing in complex networks. Then we propose a novel hybrid framework that functionally approximates trust functions defined over multi-dimensional trust tensors using both neural networks and Lie algebra-based mappings. The approximation capability of neural networks in this context and utilize Lie group structures are to capture structural symmetries and behaviors in trust propagation.

1 Introduction

In the recent years, lots of research concerns with trust modeling in complex networks have been proposed, such as [3, 5, 6, 8, 9]. However, most of work lack concerns of tensor representation of features and applying neuron network models and Lie algebra [7] for approximating trust functions. This paper is a continuation of our study [4] to address the research gap. The content is to describe the tensor representation of trust model and then to apply the functional approximate [1, 2] for trust functions.

Key words: trust modeling, lie algebra, approximation, tensor model, deep learning.
2020 AMS Mathematics classification: 68T07, 68T99, 91D30, 68R10, 68M10.

2 General Definition in Tensor

A complex network is modeled as a directed graph $G = (V, E)$, in which V is a set of nodes (agents) and E is a set of edges. In this paper, we denote n to be the number of nodes in V. Let $\mathcal{T} \in [0,1]^{n \times n \times d}$ be a 3-way trust tensor, where $\mathcal{T}_{i,j,:} = \boldsymbol{x}(i,j) \in [0,1]^d$ denotes the normalized feature vector characterizing the trust-related interaction from node i (trustor) to node j (trustee). Each slice $\boldsymbol{x}(i,j)$ includes d normalized factors such as familiarity, responsibility, dispatching efficiency, influence, etc. In trust-aware systems and multi-agent networks, a tensor-based representation enables multidimensional modeling of trust relationships. We formally define the trust tensor as follows.

Definition 1 (Trust Tensor)**.** *Let $\mathcal{T} \in [0,1]^{n \times n \times d}$ be a third-order tensor, where:*

- *n is the number of nodes (agents) in the network,*
- *d is the number of trust-related features,*
- *$\mathcal{T}_{i,j,:} = \boldsymbol{x}(i,j) \in [0,1]^d$ is the trust feature vector from node i (trustor) to node j (trustee).*

Each entry $\mathcal{T}_{i,j,k} \in [0,1]$ represents the normalized value of the k-th trust factor characterizing the interaction between i and j. Thus, the trust tensor is a mapping:

$$\mathcal{T} : \{1, \ldots, n\} \times \{1, \ldots, n\} \to [0,1]^d, \quad (i,j) \mapsto \boldsymbol{x}(i,j).$$

For example, **Figure 1** illustrates a tensor of relationships from a node (trustor) to another (trustee) with three features.

2.1 Example Trust Tensor

Consider a network with such $n = 3$ nodes and $d = 4$ trust features. It means each trust vector $\boldsymbol{x}(i,j) \in [0,1]^d$ typically includes the following components:

- $x_1(i,j)$: familiarity from i to j,
- $x_2(i,j)$: responsibility of j as perceived by i,
- $x_3(i,j)$: dispatching efficiency from i to j,
- $x_4(i,j)$: influence of node j in the global or local network.

The trust feature vector from node 1 to node 2 is:

$$\boldsymbol{x}(1,2) = \mathcal{T}_{1,2,:} = [0.8,\ 0.6,\ 0.9,\ 0.4]\,,$$

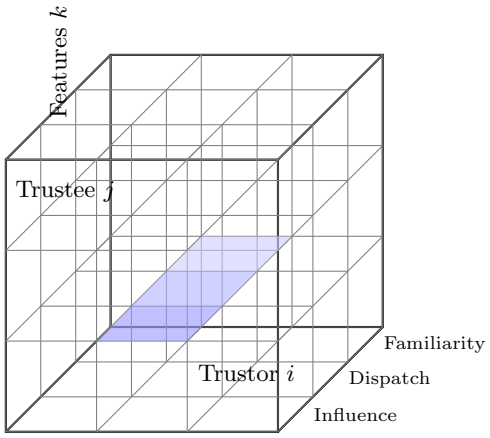

Figure 1: 3D Trust Tensor $\mathcal{T}(i, j, k)$ — capturing trustor-trustee-feature relationships.

and from node 2 to node 3 is:

$$\boldsymbol{x}(2,3) = \mathcal{T}_{2,3,:} = [0.3,\ 0.7,\ 0.5,\ 0.6]\,.$$

Here, the vector entries correspond respectively to *familiarity*, *responsibility*, *dispatch efficiency*, and *influence*, normalized into the interval $[0, 1]$. This tensor structure forms the input for neural models (e.g., CNNs, LSTMs), enabling learning over multidimensional, directional trust representations.

3 Trust Function on Tensor

Definition 2 (Trust Function in Tensor Space)**.** *A* trust function *is a mapping*

$$\mathcal{F}_{\text{trust}} : [0,1]^d \to [0,1], \quad \textit{with} \ \ \mathcal{F}_{\text{trust}}(\mathcal{T}_{i,j,:}) = \text{trust}(i,j),$$

that satisfies the following conditions:

(C1) Boundedness: *For all $\boldsymbol{x} \in [0,1]^d$, we have $\mathcal{F}_{\text{trust}}(\boldsymbol{x}) \in [0,1]$.*

(C2) Continuity: *The function $\mathcal{F}_{\text{trust}}$ is continuous on $[0,1]^d$.*

(C3) Differentiability: *$\mathcal{F}_{\text{trust}}$ is continuously differentiable on the open unit cube $(0,1)^d$, i.e., $\mathcal{F}_{\text{trust}} \in \mathcal{C}^1((0,1)^d)$.*

(C4) Monotonicity (Optional): *If a feature x_k positively contributes to trust, then*

$$\frac{\partial \mathcal{F}_{\text{trust}}}{\partial x_k}(\boldsymbol{x}) \geq 0 \quad \textit{for all } \boldsymbol{x} \in (0,1)^d.$$

This tensor-based formulation enables generalized trust modeling in multi-relational networks. It is particularly suitable for integration with deep learning models (e.g., tensorized neural architectures), differentiable optimization, and advanced trust propagation mechanisms across heterogeneous graph data.

3.1 Properties of Trust Function Class

Proposition 1 (Convexity). *Let $\mathcal{F}_{\text{trust}} : [0,1]^d \to [0,1]$ satisfy (C1)-(C3).*

(a) If $\mathcal{F}_{\text{trust}}$ is convex, then any weighted average of trust vectors results in a trust score no worse than the average trust:

$$\mathcal{F}_{\text{trust}}(\lambda \boldsymbol{x}_1 + (1-\lambda)\boldsymbol{x}_2) \leq \lambda \mathcal{F}_{\text{trust}}(\boldsymbol{x}_1) + (1-\lambda)\mathcal{F}_{\text{trust}}(\boldsymbol{x}_2), \quad \forall \lambda \in [0,1].$$

(b) If $\mathcal{F}_{\text{trust}}$ is Lipschitz continuous with constant L, then:

$$|\mathcal{F}_{\text{trust}}(\boldsymbol{x}) - \mathcal{F}_{\text{trust}}(\boldsymbol{y})| \leq L\|\boldsymbol{x} - \boldsymbol{y}\|_2 \quad \forall \boldsymbol{x}, \boldsymbol{y} \in [0,1]^d.$$

4 Approximation of Trust Functions

Theorem 1 ([1, 2] Universal Approximation Theorem). *Let $\sigma : \mathbb{R} \to \mathbb{R}$ be any non-constant, bounded, and continuous activation function. Then, for any continuous function $f : [0,1]^n \to \mathbb{R}$ and for any $\varepsilon > 0$, there exists a feedforward neural network with a single hidden layer of the form:*

$$F(\boldsymbol{x}) = \sum_{i=1}^{N} \alpha_i \, \sigma(\boldsymbol{w}_i^\top \boldsymbol{x} + \theta_i),$$

where $N \in \mathbb{N}$, $\alpha_i \in \mathbb{R}$, $\boldsymbol{w}_i \in \mathbb{R}^n$, and $\theta_i \in \mathbb{R}$, such that:

$$\sup_{\boldsymbol{x} \in [0,1]^n} |f(\boldsymbol{x}) - F(\boldsymbol{x})| < \varepsilon.$$

4.1 Approximation with Neuron Network Models

Proposition 2 (Neural Approximability of Trust Functions). *Let $\mathcal{F}_{\text{trust}} : [0,1]^d \to [0,1]$ be any continuous trust function. Then, for any $\varepsilon > 0$, there exists a feedforward neural network $\hat{\mathcal{F}}$ with a single hidden layer and sigmoidal activation such that*

$$\sup_{\boldsymbol{x} \in [0,1]^d} \left| \mathcal{F}_{\text{trust}}(\boldsymbol{x}) - \hat{\mathcal{F}}(\boldsymbol{x}) \right| < \varepsilon.$$

Proof. This is a direct application of the Universal Approximation Theorem 1. Since $\mathcal{F}_{\text{trust}}$ is continuous on a compact domain $[0,1]^d$, a neural network with a single hidden layer and non-constant bounded continuous activation (e.g., sigmoid or ReLU) can approximate it arbitrarily well. □

Proposition 3 (ReLU-CNN Approximability)**.** *Any trust function $\mathcal{F}_{trust}$ satisfying conditions (C1)-(C3) can be approximated arbitrarily well by a CNN with ReLU activation.*

Proof. By the Universal Approximation Theorem (UAT), any continuous function on a compact domain (here $[0,1]^d$) can be approximated by a feedforward neural network with ReLU. CNNs with ReLU act as structured feedforward networks with local filters, and can approximate $\mathcal{F}_{\text{trust}}$ using stacked convolutional layers and dense outputs:

$$\|\mathcal{F}_{\text{trust}}(\boldsymbol{x}) - \mathcal{F}_{\text{CNN}}(\boldsymbol{x})\|_{\infty} < \epsilon$$

□

4.2 Function Approximation on Lie Algebras

Let $\mathfrak{g}$ be a finite-dimensional real Lie algebra. Since every finite-dimensional Lie algebra $\mathfrak{g}$ is a real vector space of dimension d, we have $\mathfrak{g} \cong \mathbb{R}^d$ as topological vector spaces. Therefore, any continuous function defined on $\mathfrak{g}$ can be approximated using a standard feedforward neural network.

Proposition 4 (Universal Approximation on Lie Algebras)**.** *Let $\mathfrak{g}$ be a finite-dimensional Lie algebra over $\mathbb{R}$, and let $f : \mathfrak{g} \to \mathbb{R}$ be a continuous function. Then, for any compact subset $K \subset \mathfrak{g}$ and any $\varepsilon > 0$, there exists a neural network $\hat{f} : \mathfrak{g} \to \mathbb{R}$ with a single hidden layer such that:*

$$\sup_{X \in K} \left| f(X) - \hat{f}(X) \right| < \varepsilon.$$

This result follows from the classical Universal Approximation Theorem applied to $\mathbb{R}^d$, since any Lie algebra $\mathfrak{g}$ of dimension d can be identified with $\mathbb{R}^d$. In practice, we can flatten the matrix representation of an element $X \in \mathfrak{g}$ (e.g., skew-symmetric or traceless matrix) into a vector $\boldsymbol{x} \in \mathbb{R}^d$, and train a neural network $\hat{f}_\theta(\boldsymbol{x})$ to approximate $f(X)$.

4.3 Illustrated Example

Let $\mathfrak{g} = \mathfrak{so}(3)$ and denote a skew-symmetric matrix by

$$X = \begin{bmatrix} 0 & -x_3 & x_2 \\ x_3 & 0 & -x_1 \\ -x_2 & x_1 & 0 \end{bmatrix}, \qquad \mathbf{x} = (x_1, x_2, x_3)^\top \in \mathbb{R}^3.$$

Suppose the trust function depends on the Lie-algebra feature X (e.g. local rotational influence) and scalar features $\mathbf{s} = (\text{fam}, \text{resp}, \text{disp}) \in [0,1]^3$. Define

$$f : \mathfrak{so}(3) \times [0,1]^3 \to [0,1], \qquad f(X, \mathbf{s})$$

as any continuous scoring rule (e.g. a normalized utility). By Proposition 2, for every $\varepsilon > 0$ there exists a neural network $\widehat{F}$ taking input $\big(\mathbf{x}, \text{fam}, \text{resp}, \text{disp}\big) \in \mathbb{R}^6$ that approximates f uniformly on any compact subset.
For more structured learning tasks on Lie algebras or Lie groups, geometric deep learning and equivariant neural networks can be employed to better respect the algebraic properties. Figure 2 illustrates a Trust Tensor Slice $\mathcal{T}_{i,j,:}$ and Neural Network Approximator.

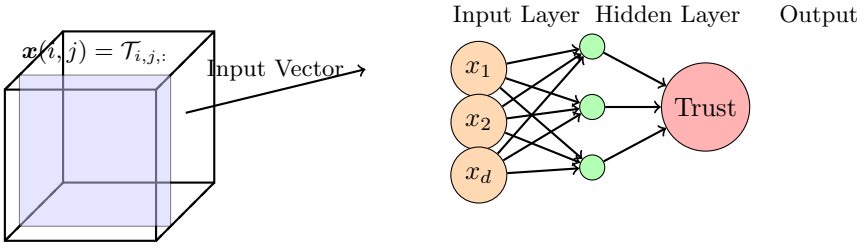

Figure 2: Trust Tensor Slice $\mathcal{T}_{i,j,:}$ and Neural Network Approximator

5 Conclusion and Discussions

This paper describes the tensor concept for modeling trust computing and trust functions on tensors. Some results on functional approximation have been presented. The approximation theorem allows approximating f by a network after flattening, it is often advantageous to respect Lie structure:

- If f is invariant or equivariant under the adjoint action of a Lie group (or other group symmetries), impose those symmetries in the architecture (e.g. use layers that are invariant under conjugation or the adjoint action).
- Use exponential/log maps when combining Lie-algebra features with group-valued features, or parameterize outputs on the Lie algebra to preserve algebraic constraints.
- Graph neural networks or message-passing models that incorporate Lie features (as edge/node features) can capture relational structure in trust networks.

However, when modelling in practice, we need:

1. Choose an identification $\Phi : \mathfrak{g} \to \mathbb{R}^d$ (e.g. vectorization of matrix entries or a basis expansion). Standardize/normalize features on a compact domain (helps approximation and training).

2. If invariances are known (e.g. invariance under certain group actions), design equivariant layers. Train the network on labeled trust-data $(X_i, \mathbf{s}_i, y_i)$ with a loss such as MSE or cross-entropy, depending on the target.

3. Validate on held-out data and, if needed, restrict model to lie-algebra-respecting outputs (project back to $\mathfrak{g}$ with linear constraints). The universal result is *existential* — it does not give the number of neurons nor a training procedure.

4. If $\mathfrak{g}$ is high-dimensional, sample complexity may be large; structure-aware models reduce sample needs. For time-varying or stochastic trust, consider dynamic models (RNNs, continuous-time flows) and extend the compactness/continuity assumptions accordingly.

These probelms need to be investigated furthermore. The research results will be presented in our future work.

References

[1] Cybenko George, Approximation by super,positions of a sigmoidal function, Mathematics of Control, Signals and Systems, Vol.2, No.4, 303–314, Springer, 1989

[2] Hornik, Kurt and Stinchcombe et alt., Multilayer feedforward networks are universal approximators, Journal of Neural Networks, Vol.2, No.5, pp.359–366, Pergamon, 1989.

[3] Dinh Que Tran, Phuong Thanh Pham, TreeXTrust: A Tree-Based Explainable Trust Model Using Social Semantics and Influence, *Journal of Computer Science*, AGH University of Science and Technology, 26(2), pp. 1–25, 2025.

[4] Dinh Que Tran, Initial study on trust computation based on tensor and lie algebra in complex networks, submitted to Southeast Asian Journal of Sciences.

[5] Zheng Du, Min-Hung Chen, and Yung-Hsiang Lu, Tensor-based Trust Evaluation with Lie Algebra for Dynamic Networks, *Journal of Computer Science and Systems*, vol. 19(3), 2023.

[6] Taco Cohen and Max Welling, Group Equivariant Convolutional Networks, *International Conference on Machine Learning (ICML)*, 2016.

[7] Alexander Kirillov, Introduction to Lie Groups and Lie Algebras, Cambridge University Press, 2008.

[8] Mei Lu and Fanzhang Li, Survey on Lie Group Machine Learning, *https://www.sciopen.com/article/10.26599/BDMA.2020.9020011*, 2020.

[9] Sheng Zhang, Hang Su, and Jun Zhu, Learning Graphical Lie Algebra with Deep Structured Embedding, *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.