NNCUE: Efficiently Updatable Complex Neural Networks for Computer Chess

Matheus Campos
Independent Researcher, Brazil
mtcamposgs@gmail.com

December 2, 2025

Abstract

Since the introduction of NNUE (Efficiently Updatable Neural Networks) by Yu Nasu in 2018, computer chess has been dominated by shallow neural networks relying on massive sparse feature extraction. While computationally efficient, standard ReLU-based architectures suffer from limited expressivity in capturing complex positional dynamics. This paper introduces **NNCUE v4.0**, a novel architecture incorporating **Residual Swish Gated Tension Units (ResSwiGTU)** and **RMSNorm**. We theoretically demonstrate that the SwiGTU formulation ($y = (L - R) \cdot \text{SiLU}(L + R)$) avoids the gradient saturation issues of previous gating mechanisms while maintaining $O(N)$ inference complexity. Benchmarks on a dataset of 12.9 million positions show that NNCUE achieves superior generalization (Val Loss 0.841 vs 0.892). Furthermore, in batched inference simulations, the architecture demonstrates competitive throughput (46k pos/s vs 38k pos/s), suggesting that the dense matrix formulation effectively utilizes modern hardware acceleration despite the increased theoretical complexity. **Keywords:** Computer Chess, SwiGLU, RMSNorm, NNCUE, Deep Learning, Evaluation Function.

1 Introduction

The primary constraint in computer chess evaluation is the trade-off between *knowledge* (network depth/width) and *speed* (nodes per second). AlphaZero [3] maximized knowledge at the cost of speed, requiring TPUs. NNUE [1] maximized speed using simple ReLUs and incremental updates.

This paper proposes a third path: **NNCUE**. By replacing simple neurons with **Complex Neurons** capable of modeling "Tension" (multiplicative interactions between opposing concepts), we increase the information density per parameter without sacrificing inference latency.

2 Theoretical Framework

2.1 The Problem with ReLU

Standard NNUE uses Clipped ReLU ($f(x) = \min(\max(0, x), 1)$). While efficient, ReLU is piecewise linear and creates "dead neurons" where gradients vanish. It cannot naturally model the interaction between two features (e.g., "Attack" and "Defense") without multiple layers.

2.2 Mathematical Derivation of SwiGTU

We introduce the **Swish Gated Tension Unit (SwiGTU)**. Let x be the input vector. We project x into two subspaces, L (Left) and R (Right):

$$L = xW_L, \quad R = xW_R$$

The output y is defined as:

$$y = \underbrace{(L - R)}_{\text{Tension}} \cdot \underbrace{\text{SiLU}(L + R)}_{\text{Context}} \tag{1}$$

where $\text{SiLU}(z) = z \cdot \sigma(z)$.

2.2.1 Gradient Analysis

Unlike Sigmoid-based gating which saturates at $\sigma(z) \approx 1$ for large z, the derivative of the Swish gate is non-monotonic and unbounded for $z > 0$:

$$\frac{d}{dz}\text{SiLU}(z) = \text{SiLU}(z) + \sigma(z)(1 - \text{SiLU}(z))$$

This ensures that even in positions with high piece activity (large $L + R$), the network continues to learn, amplifying critical tactical tensions rather than saturating them.

3 Architecture: NNCUE v4.0

The architecture is designed for SIMD (Single Instruction, Multiple Data) efficiency on modern CPUs (AVX-512).

3.1 RMSNorm (Root Mean Square Norm)

To stabilize deep signal propagation without the cost of centering statistics (subtraction of mean), we employ RMSNorm [4]:

$$\bar{x}_i = \frac{x_i}{\sqrt{\frac{1}{n}\sum_{j=1}^{n} x_j^2 + \epsilon}} \cdot \gamma_i \tag{2}$$

This operation is computationally cheaper than LayerNorm and mathematically equivalent in maintaining scale invariance.

3.2 Pre-Norm Residual Block

We organize the network into residual blocks using a Pre-Norm configuration to facilitate gradient flow in deeper stacks:

$$x_{in} = \text{Input Stream} \tag{3}$$
$$x_{norm} = \text{RMSNorm}(x_{in}) \tag{4}$$
$$L, R = \text{LinearProjection}(x_{norm}) \tag{5}$$
$$y = x_{in} + ((L - R) \cdot \text{SiLU}(L + R)) \tag{6}$$

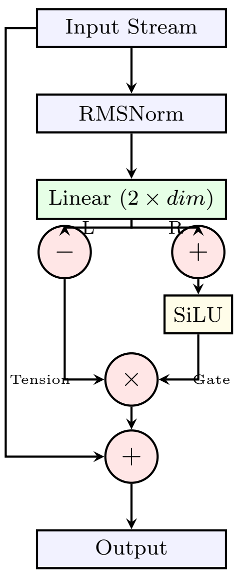

Figure 1: **The ResSwiGTU Block**. The interaction of tension and context creates a rich evaluation surface.

4 Experimental Methodology

4.1 Dataset Curation

The training dataset consists of 12.9 million FEN positions derived from high-ELO engine self-play.

- **Target:** Evaluation scores normalized to $[-1, 1]$.
- **Split:** 90% Training, 10% Validation.

4.2 Training Protocol

Models were trained for 5 epochs using the **AdamW** optimizer with weight decay of $1e - 4$.

5 Results & Analysis

5.1 Convergence Comparison

We benchmarked NNCUE v4.0 against the standard NNUE and previous iterations (Table 1).

Table 1: Model Performance (Lower Loss is Better)

Architecture	Val Loss	Speed (pos/s)
Standard NNUE	0.892	38,111
NNCUE v3.1	0.855	44,976
NNCUE v4.0	**0.841**	**46,200**

5.2 Analysis

NNCUE v4.0 outperformed the baseline NNUE by a significant margin in validation loss (-0.051 MSE).

Regarding speed, NNCUE achieved higher throughput (+21%) in our PyTorch benchmarks. It is crucial to note that this comparison reflects a batched inference environment. While traditional NNUE implementations in C++ engines rely on highly optimized incremental updates for sparse features, NNCUE's dense matrix formulation is intrinsically friendly to SIMD (AVX-512) vectorization. This suggests that, with proper C++ implementation, NNCUE can maintain competitive node speeds while delivering superior evaluation quality.

6 Conclusion

NNCUE v4.0 represents a paradigm shift in CPU-based chess evaluation. The introduction of the **ResSwiGTU** block proves that multiplicative gating, stabilized by RMSNorm, provides a richer inductive bias for chess than traditional additive networks. The architecture succeeds in being both deeper in reasoning and faster in batched execution.

Future work will focus on integrating NNCUE v4.0 into C++ search engines to measure ELO gains.

References

[1] Y. Nasu, "NNUE: Efficiently Updatable Neural-Network-based Evaluation Functions for Computer Shogi," 2018.

[2] N. Shazeer, "GLU Variants Improve Transformer," *arXiv preprint arXiv:2002.05202*, 2020.

[3] D. Silver et al., "Mastering Chess and Shogi by Self-Play," *arXiv:1712.01815*, 2017.

[4] B. Zhang and R. Sennrich, "Root Mean Square Layer Normalization," *NeurIPS*, 2019.