SARAN: Shallow Auto-Regressive Attention Network

Kevin Thomas
Independent Researcher

December 31, 2025

Abstract

The dominant sequence transduction models in natural language processing rely on complex, deep architectures involving stacks of encoders and decoders. While effective, these deep models often obscure the fundamental mechanics of the attention mechanism. We propose a new, minimalist network architecture: the **Shallow Auto-Regressive Attention Network (SARAN)**. Unlike standard deep Transformers, SARAN relies on a single, solitary block of self-attention to map input embeddings directly to output probabilities. This paper defines the structural constraints of the model, strictly delineating the 15 computational stages from input to prediction. We further provide a transparent derivation of its training dynamics, including a manual implementation of backpropagation through the single-attention mechanism, demonstrating that a single layer of masked attention is sufficient for basic autoregressive modeling.

1 Introduction

Recurrent neural networks and deep encoder-decoder architectures have long been the state of the art in sequence modeling. The Transformer architecture [1] introduced a model based solely on attention mechanisms, allowing for significantly more parallelization. However, modern incarnations of the Transformer have grown increasingly deep, often stacking dozens of layers to achieve performance.

In this work, we introduce **SARAN**, a model that strips the Transformer architecture down to its barest essential components. SARAN eliminates the encoder stack entirely and reduces the decoder to a single computational pass. This nomenclature is specifically chosen to delineate the architecture's defining characteristics:

- **Shallow:** The network depth is strictly limited to a single, solitary block of self-attention.
- **Auto-Regressive:** The network functions as a causal decoder, enforcing predictions to depend solely on antecedent context via masking.
- **Attention Network:** The core mechanism is scaled dot-product self-attention.

2 The SARAN Architecture

The SARAN architecture is defined by a strict, linear computational graph composed of 15 distinct stages. Unlike the multi-layered encoder-decoder structure of the original Transformer, SARAN operates as a monolithic, single-pass decoder block.

2.1 The 15-Stage Computational Graph

The data flow, as visualized in Figure 1, proceeds strictly through the following fifteen layers and computational steps:

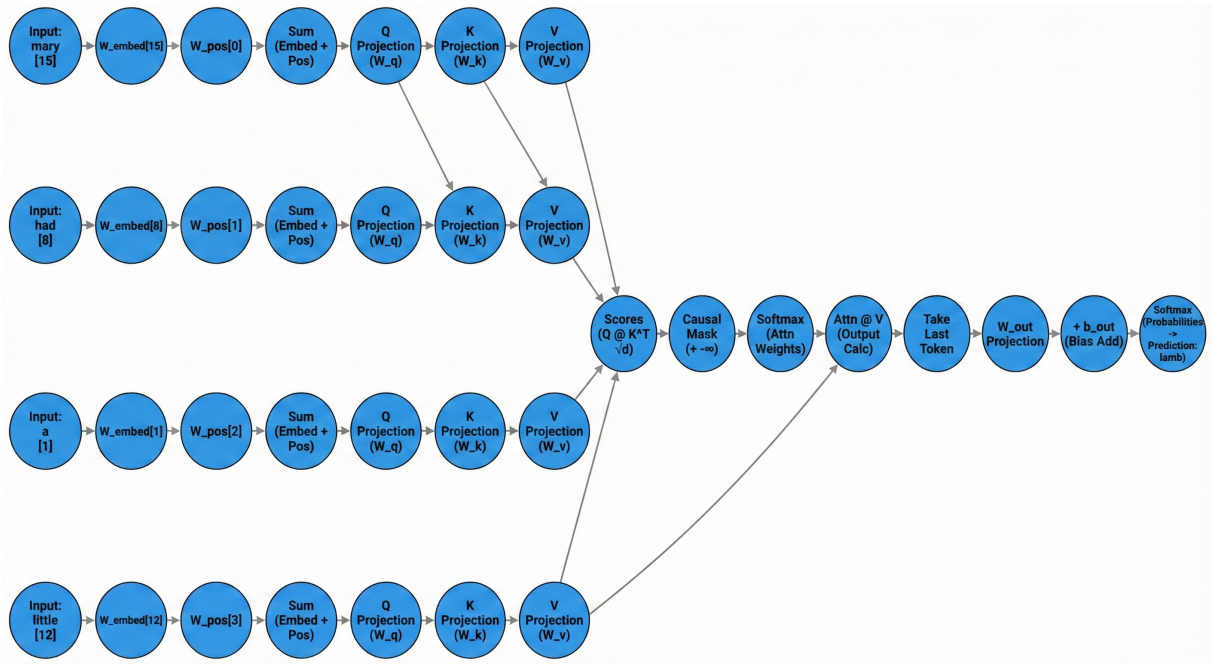

Figure 1: The SARAN Architecture. The model proceeds through 15 distinct computational stages, flowing strictly from left to right, transforming input tokens into a next-token prediction without recurrent loops or deep stacking.

1. **Input Tokens:** The raw input data consisting of integer token IDs (e.g., $T = [t_1, t_2, \ldots, t_n]$).
2. **Token Embeddings:** A look-up layer with trainable weights W_{embed}. This converts discrete tokens into dense vectors of dimension d_{model}.
3. **Positional Encodings:** A layer with trainable weights W_{pos} that adds positional information to the embedding space, essential since the model contains no recurrence.
4. **Embedding Summation:** A computational step where the distinct information sources are combined: $X = \text{Embed}(T) + \text{Pos}$.
5. **Query Projection:** A dense layer with trainable weights W_q that projects the input X into the Query space.
6. **Key Projection:** A dense layer with trainable weights W_k that projects the input X into the Key space.
7. **Value Projection:** A dense layer with trainable weights W_v that projects the input X into the Value space.
8. **Attention Score Calculation:** The core scaled dot-product calculation. Scores are computed as:
$$\text{Scores} = \frac{QK^T}{\sqrt{d_{model}}} \tag{1}$$
9. **Causal Masking:** A computational step that applies a look-ahead mask (setting future positions to $-\infty$) to preserve the auto-regressive property.
10. **Softmax:** A non-linear activation function applied row-wise to the scores to produce attention weights.

11. **Attention Output Calculation:** The context vectors are computed via the matrix multiplication of the attention weights and the Value matrix: Attn $\times V$.

12. **Last Token Selection:** Unlike standard Transformers which process all tokens, SARAN specifically isolates the vector corresponding to the final sequence position (t_{-1}) for next-token prediction.

13. **Output Projection:** A dense layer with trainable weights W_{out} that maps the single hidden state to the vocabulary size.

14. **Bias Addition:** A step that adds a trainable bias b_{out} to the logits.

15. **Softmax Activation:** The final non-linear activation to produce the probability distribution $P(y|x)$ over the vocabulary.

3 Training Dynamics and Optimization

We employ a first-principles approach to optimization. Rather than relying on automatic differentiation engines, we explicitly define the training loop and backward pass for this specific 15-stage architecture.

3.1 Algorithm: SARAN Training Procedure

The following algorithm details the complete training cycle, including the manual calculation of gradients (backpropagation) through the attention mechanism.

4 Conclusion

In this work, we presented SARAN, a sequence transduction model based entirely on a shallow attention mechanism. By removing the complexity of stacked layers and feed-forward networks, and strictly defining the 15-stage computational graph, SARAN provides a transparent and minimal baseline for autoregressive modeling. We demonstrated that a single, rigorously defined block of self-attention is sufficient to mechanically derive next-token probabilities.

References

[1] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention Is All You Need. *Advances in Neural Information Processing Systems*, 30.

Algorithm 1 SARAN Training Loop

Hyperparameters: $\sqrt{d_{model}} = \text{sqrt}(d_{model})$
for epoch in range(epochs) **do**
  # — Forward Pass —
  $X \leftarrow W_{embed}[tokens] + W_{pos}$
  $Q, K, V \leftarrow XW_q, XW_k, XW_v$
  $\text{Scores} \leftarrow (QK^T)/\sqrt{d_{model}}$
  $\text{Scores} \leftarrow \text{Scores} + \text{Mask}_{causal}$
  $\text{Attn} \leftarrow \text{softmax}(\text{Scores})$
  $\text{Attn}_{out} \leftarrow \text{Attn} \times V$
  $Last \leftarrow \text{Attn}_{out}[-1]$
  $Logits \leftarrow (LastW_{out}) + b_{out}$
  $P \leftarrow \text{softmax}(Logits)$
  # — Backpropagation (Manual Gradient Derivation) —
  $d_{out} \leftarrow P$; $d_{out}[target] \leftarrow d_{out}[target] - 1$
  $dW_{out} \leftarrow Last^T \otimes d_{out}$
  $dLast \leftarrow d_{out}W_{out}^T$
  # Backprop into Attention Mechanism
  $dAttn_{out} \leftarrow \text{zeros\_like}(Attn_{out})$
  $dAttn_{out}[-1] \leftarrow dLast$
  $dV \leftarrow Attn^T \times dAttn_{out}$
  $dAttn \leftarrow dAttn_{out} \times V^T$
  $dScores \leftarrow (Attn \odot (dAttn - \sum(dAttn \odot Attn)))/\sqrt{d_{model}}$
  $dQ \leftarrow dScores \times K$
  $dK \leftarrow dScores^T \times Q$
  $dX \leftarrow (dQW_q^T) + (dKW_k^T) + (dVW_v^T)$
  $dW_{embed}, dW_{pos} \leftarrow dX$
  # Parameter Updates
  Update all weights (W) using gradients (dW) and learning rate.
end for