Architecting LLMs

Building a GPT like model from scratch

Jovita Bhasin

January 29, 2026

Contents

1 Introduction

Large Language Models (LLMs) form the backbone of modern natural language processing systems, powering applications ranging from machine translation to conversational agents. Despite their widespread use, the internal mechanisms of these models are often obscured by high-level abstractions provided by deep learning frameworks.

This project aims to demystify the architecture and training of language models by building them incrementally from first principles. Beginning with simple probabilistic models and progressively introducing neural networks, normalization techniques, and hierarchical architectures, the project emphasizes both theoretical understanding and hands-on implementation.

The work documented here covers the design, implementation, and training of multiple language modeling approaches, culminating in a hierarchical WaveNet-style model capable of leveraging larger context windows while achieving improved validation performance.

2 Dataset and Problem Formulation

The dataset used throughout this project is `names.txt`, a collection of lowercase names. The task is framed as a **character-level language modeling problem**, where the goal is to predict the next character given a fixed-length context of preceding characters.

Let the vocabulary be denoted as:

$$\mathcal{V} = \{c_1, c_2, \ldots, c_V\}$$

including a special start/end token '.'.

Given a context window of size k, the model learns the conditional probability:

$$P(x_t \mid x_{t-k}, \ldots, x_{t-1})$$

Training data is generated using a sliding window mechanism, and the dataset is split into training, validation, and test partitions to evaluate generalization.

3 Bigram Language Model

The simplest model implemented is a **bigram language model**, where the probability of the next character depends only on the immediately preceding character.

$$P(x_t \mid x_{t-1})$$

A count matrix $C \in \mathbb{R}^{V \times V}$ is constructed, where C_{ij} represents the number of times character j follows character i. After applying smoothing, probabilities are obtained via row-wise normalization:

$$P_{ij} = \frac{C_{ij} + \alpha}{\sum_j (C_{ij} + \alpha)}$$

This model serves as a baseline and introduces the core idea of likelihood-based training.

4 Neural Bigram Model

The bigram model is reinterpreted as a neural network by parameterizing the logits using a weight matrix W:

$$\text{logits} = W[x_{t-1}]$$

The probability distribution is obtained using the softmax function:

$$P(x_t = j) = \frac{e^{\text{logit}_j}}{\sum_k e^{\text{logit}_k}}$$

Training is performed by minimizing the cross-entropy loss:

$$\mathcal{L} = -\frac{1}{N} \sum_{i=1}^{N} \log P(y_i \mid x_i)$$

This formulation enables gradient-based optimization and establishes the foundation for deeper neural models.

5 Multi-Layer Perceptron Language Model

To incorporate larger context windows, a multi-layer perceptron (MLP) is introduced. Each character in the context is mapped to a learned embedding:

$$\mathbf{e}_i = E[x_i]$$

The embeddings are concatenated and passed through a hidden layer:

$$\mathbf{h} = \tanh(\mathbf{W}_1[\mathbf{e}_1; \ldots; \mathbf{e}_k] + \mathbf{b}_1)$$

The output logits are computed as:

$$\text{logits} = \mathbf{W}_2\mathbf{h} + \mathbf{b}_2$$

This architecture significantly improves modeling capacity but introduces optimization challenges as depth and width increase.

6 Batch Normalization

Batch normalization is applied to stabilize training by normalizing hidden pre-activations:

$$\mu = \frac{1}{n}\sum_{i=1}^{n} x_i, \quad \sigma^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \mu)^2$$

The normalized activations are:

$$\hat{x} = \frac{x - \mu}{\sqrt{\sigma^2 + \epsilon}}$$

Followed by learnable scale and shift parameters:

$$y = \gamma\hat{x} + \beta$$

Manual backpropagation through batch normalization was implemented to understand gradient flow through normalization layers.

7 Manual Backpropagation and Autograd Verification

A major component of the project involved manually deriving and implementing backward passes for complex computational graphs, including softmax, cross-entropy loss, and batch normalization.

For softmax-cross-entropy, the gradient simplifies to:

$$\frac{\partial \mathcal{L}}{\partial \text{logits}} = \text{softmax}(\text{logits}) - y_{\text{one-hot}}$$

Each manually computed gradient was verified against PyTorch's automatic differentiation using numerical comparison, ensuring correctness and deepening understanding of gradient propagation.

8 Detailed Gradient Derivations

To develop a rigorous understanding of learning dynamics in neural language models, explicit gradient derivations were carried out for all major components of the computational

graph.

8.1 Cross-Entropy Loss with Softmax

Given logits $\mathbf{z} \in \mathbb{R}^V$, the softmax function is defined as:

$$\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_j e^{z_j}}$$

For a one-hot encoded target vector $\mathbf{y}$, the cross-entropy loss is:

$$\mathcal{L} = -\sum_i y_i \log \sigma(\mathbf{z})_i$$

Differentiating with respect to z_k yields:

$$\frac{\partial \mathcal{L}}{\partial z_k} = \sigma(\mathbf{z})_k - y_k$$

This elegant simplification was explicitly verified in code and forms the basis for efficient gradient computation in classification models.

8.2 Batch Normalization Backward Pass

Batch normalization introduces dependencies across samples in a batch, making its backward pass non-trivial. Given:

$$\hat{x}_i = \frac{x_i - \mu}{\sqrt{\sigma^2 + \epsilon}} \quad \text{and} \quad y_i = \gamma \hat{x}_i + \beta$$

The gradients must propagate through mean and variance computations:

$$\frac{\partial \mathcal{L}}{\partial x_i} = \frac{1}{N}\gamma(\sigma^2 + \epsilon)^{-1/2}\left(N\frac{\partial \mathcal{L}}{\partial y_i} - \sum_j \frac{\partial \mathcal{L}}{\partial y_j} - \hat{x}_i \sum_j \frac{\partial \mathcal{L}}{\partial y_j}\hat{x}_j\right)$$

Manually implementing and validating this expression reinforced an understanding of gradient aggregation and broadcasting effects.

9 Hierarchical WaveNet-Style Architecture

To move beyond flat MLPs, a hierarchical architecture inspired by WaveNet was implemented. Instead of concatenating all context embeddings at once, the model progressively fuses neighboring representations.

For a context size of 8, the hierarchy proceeds as:

$$8 \to 4 \to 2 \to 1$$

At each stage, pairs of vectors are concatenated and passed through a shared transformation block:

$$\mathbf{h}_i^{(l+1)} = f([\mathbf{h}_{2i}^{(l)}; \mathbf{h}_{2i+1}^{(l)}])$$

This design encourages the model to learn local patterns before composing higher-level abstractions.

10 Architectural Design Analysis

The progression from flat multi-layer perceptrons to hierarchical architectures reflects a fundamental shift in how contextual information is modeled.

10.1 Flat vs Hierarchical Representations

In flat MLP models, all contextual embeddings are concatenated and processed simultaneously:

$$[\mathbf{e}_1; \mathbf{e}_2; \ldots; \mathbf{e}_k]$$

While expressive, this approach scales poorly with increasing context size and places a heavy burden on a single transformation layer.

In contrast, hierarchical fusion decomposes the modeling task into stages, each responsible for combining increasingly abstract representations. This mirrors principles found in convolutional neural networks, where receptive fields grow with depth.

10.2 Parameter Efficiency

Hierarchical models reuse the same transformation block at each fusion stage, significantly reducing parameter growth compared to flat architectures with large input dimensions. This enables the use of wider embeddings and deeper processing without prohibitive computational cost.

10.3 Information Flow and Residual Connections

Residual connections were critical in preserving gradient flow across hierarchical levels:

$$\mathbf{h}^{(l+1)} = f(\mathbf{h}^{(l)}) + \mathbf{h}^{(l)}$$

Without residuals, early fusion layers exhibited gradient attenuation, limiting effective depth. Residual pathways mitigated this issue and enabled stable optimization.

11 Residual Connections and Optimization Improvements

To address vanishing gradients in the hierarchical model, residual connections were added:

$$\mathbf{y} = f(\mathbf{x}) + \mathbf{x}$$

Combined with batch normalization, GELU activations, dropout, and AdamW optimization, this significantly improved convergence and generalization.

Learning rate scheduling and careful initialization further reduced validation loss below 1.9, outperforming earlier flat MLP baselines while using larger context windows.

12 Results and Evaluation

Models were evaluated using cross-entropy loss on held-out validation and test sets. The hierarchical WaveNet-style model achieved:

- Lower validation loss compared to flat MLPs
- Better utilization of extended context
- More coherent character-level samples

Qualitative inspection of generated names demonstrated improved structural consistency and reduced randomness.

13 Limitations and Failure Modes

Despite strong empirical performance, several limitations were observed.

13.1 Context Compression Loss

Hierarchical fusion inevitably compresses information as representations are merged. Although residual connections alleviate this to some extent, fine-grained positional details may still be lost at deeper levels.

13.2 Sensitivity to Optimization Hyperparameters

The hierarchical WaveNet-style model was significantly more sensitive to learning rate schedules and initialization compared to flat MLPs. Improper decay or excessive regularization often resulted in stalled training.

13.3 Scalability Considerations

While effective for character-level modeling on small datasets, this architecture does not scale efficiently to very large vocabularies or long contexts. In such regimes, attention-based models provide superior flexibility and performance.

These observations motivate the transition toward transformer architectures in subsequent work.

14 Conclusion

This project provides a bottom-up exploration of language modeling, progressing from simple probabilistic methods to structured neural architectures. By implementing models manually and validating each component, the work builds a strong conceptual foundation for understanding modern LLMs.

The hierarchical WaveNet-style model represents a critical step toward more advanced architectures such as convolutional and transformer-based language models. Future work may extend this framework to causal convolutions, attention mechanisms, and large-scale training regimes.