Computational Analysis of Elman RNN on enwik9
Baseline Performance and Optimization Targets

Claude

2026-02-06

Abstract

We analyze the computational requirements for running an Elman RNN on the full enwik9 dataset (1 billion bytes). The current single-threaded C implementation achieves approximately 2,800 bytes/second, requiring ∼100 hours for a single forward pass. We characterize the bottlenecks and establish optimization targets for practical tick-tock iteration.

1 Introduction

The tick-tock methodology alternates between training (tick) and interpretation (tock). Each tick requires multiple passes over enwik9—at minimum one forward pass for evaluation, and many more for gradient-based training. At 100 hours per pass, a single training epoch takes 4 days, making rapid iteration impossible.

This paper establishes the baseline computational profile and identifies optimization opportunities.

2 Model Architecture

The Elman RNN has architecture $256 \to 128 \to 256$:

$$h_t = \tanh(W_{ih}x_t + W_{hh}h_{t-1} + b_h) \tag{1}$$

$$y_t = \text{softmax}(W_{ho}h_t + b_o) \tag{2}$$

Parameters:

Matrix	Dimensions	Parameters
W_{ih}	128×256	32,768
W_{hh}	128×128	16,384
b_h	128	128
W_{ho}	256×128	32,768
b_o	256	256
Total		82,304

Model size: 329 KB (float32).

3 Operations per Timestep

For each of the 1 billion input bytes:

Operation	Computation	FLOPs
$W_{ih}x_t$ (sparse)	column lookup	256
$W_{hh}h_{t-1}$	128×128 matmul	32,768
$+b_h$	vector add	128
tanh	128 nonlinearities	$\sim$1,280
$W_{ho}h_t$	256×128 matmul	65,536
$+b_o$	vector add	256
softmax	exp, sum, div	$\sim$2,560
cross-entropy	log lookup	$\sim$257
Total		**$\sim$103,000**

Note: The input x_t is one-hot, so $W_{ih}x_t$ reduces to a single column lookup rather than a full matrix-vector multiply.

4 Total Computation for enwik9

Metric	Value
Dataset size	1,000,000,000 bytes
FLOPs per byte	103,000
Total FLOPs	103 TFLOPs

5 Memory Analysis

5.1 Weight Matrices

All weights fit comfortably in L2/L3 cache:

Component	Size
W_{ih}	128 KB
W_{hh}	64 KB
W_{ho}	128 KB
Biases	1.5 KB
Total	321.5 KB

5.2 Per-Timestep Memory Traffic

With weights cached:

- Read: h_{t-1} (512 bytes), input byte (1 byte)
- Write: h_t (512 bytes), output logits (1 KB)
- Total: $\sim$1.5 KB per timestep
- Full dataset: $\sim$1.5 TB memory traffic

6 Baseline Performance

6.1 C Implementation (hutter.c)

Benchmarked on server CPU (single-threaded):

Sample Size	Time	Throughput
1,000 bytes	0.39 s	2,551 bytes/s
10,000 bytes	3.85 s	2,597 bytes/s
100,000 bytes	30.6 s	3,263 bytes/s

Extrapolated time for full enwik9: 99 hours (4.1 days)

6.2 Python/NumPy Implementation

Single-threaded with BLAS:

Metric	Value
Throughput	∼10,000 bytes/s
Extrapolated enwik9 time	∼28 hours

NumPy outperforms the C implementation because it uses optimized BLAS routines for the matrix-vector multiplications.

6.3 Analysis

The C implementation achieves only:

$$\frac{103,000 \text{ FLOPs}}{1/2800 \text{ s}} = 288 \text{ MFLOPs/s}$$

This is ∼3% of single-core capability (∼10 GFLOPs/s for modern x86). The bottleneck is not raw compute but rather:

1. Lack of SIMD vectorization
2. Sequential loop structure
3. Possible cache inefficiency in memory layout

7 Theoretical Limits

7.1 CPU (Single Core)

Modern x86 with AVX2: ∼100 GFLOPs/s peak, ∼10-30 GFLOPs/s sustained.

Sustained GFLOPs/s	enwik9 Time
10	2.9 hours
30	57 minutes

7.2 GPU

RNNs are memory-bound on GPU due to sequential dependency. Effective throughput is typically 1-5% of peak FLOPs.

GPU	Effective TFLOPs	enwik9 Time
RTX 3090 (35 TF peak)	0.5–1.0	2–3 minutes
A100 (312 TF peak)	3–5	20–30 seconds

7.3 Batched Processing

Processing B independent sequences in parallel:

- Matrix-vector $\rightarrow$ matrix-matrix multiplication
- Better cache utilization and SIMD efficiency
- Requires $B\times$ memory for hidden states

For batch size $B = 64$:

- Each sequence: 15.6M bytes (enwik9 / 64)
- Hidden state memory: 64 $\times$ 512 bytes = 32 KB
- Potential speedup: 10–50$\times$ on CPU, 100–1000$\times$ on GPU

8 Optimization Targets

Optimization	Expected Speedup	Target Time
Baseline (current)	1$\times$	99 hours
SIMD vectorization	5–10$\times$	10–20 hours
+ Batched (B=64)	20–50$\times$	2–5 hours
GPU (cuDNN RNN)	100–500$\times$	10–60 minutes

9 Optimization Roadmap

9.1 Phase 1: SIMD Vectorization

1. Replace scalar loops with AVX2 intrinsics
2. Ensure 32-byte aligned memory layout
3. Use `-O3 -march=native -ffast-math`
4. Target: 10$\times$ speedup (10 hours per pass)

9.2 Phase 2: Batched Processing

1. Split enwik9 into B chunks
2. Process chunks in parallel with BLAS `sgemm`
3. Hidden states: $B \times 128$ matrix
4. Target: 50× speedup (2 hours per pass)

9.3 Phase 3: GPU Acceleration

1. Port to PyTorch or cuDNN
2. Use fused RNN kernels
3. Target: 500× speedup (10 minutes per pass)

10 Results After Optimization

10.1 Phase 1 Results: SIMD Vectorization

Implemented AVX2 SIMD with 8-way loop unrolling and FMA instructions.

Compiler flags: `-O3 -march=native -mavx2 -mfma -ffast-math`

Key optimizations:

1. 8-way unrolled dot products using `_mm256_fmadd_ps`
2. Fast tanh approximation using Padé polynomial
3. Horizontal sum via `_mm256_extractf128_ps` shuffle
4. Aligned memory access for weight matrices

Results:

Version	Throughput	enwik9 Time	Speedup
Original hutter.c	2,800 steps/s	99 hours	1.0×
Scalar (-O3 -march=native)	11,800 steps/s	23.5 hours	4.2×
SIMD V3 (8x unroll + FMA)	37,400 steps/s	7.4 hours	13.3×

Analysis: The original hutter.c was compiled without optimization flags, explaining the 4.2× improvement from just `-O3 -march=native`. The SIMD implementation adds another 3.2× on top, for a total of 13.3×.

Conclusion: Phase 1 exceeded the 10× target. Single forward pass over enwik9 now takes 7.4 hours instead of 99 hours.

10.2 Phase 2 Results: OpenMP Parallelization

Implemented parallel processing of independent sequences using OpenMP. Each thread processes its own segment of enwik9 with the Phase 1 SIMD kernel.

System: AMD EPYC 7571 with 2 vCPUs (1 core, 2 threads via SMT).

Results:

Threads	Throughput	enwik9 Time	Total Speedup
1 (Phase 1 only)	34,000 bytes/s	8.2 hours	12.2×
2	42,700 bytes/s	6.5 hours	15.2×
4 (hyperthreading)	43,100 bytes/s	6.4 hours	15.4×

Analysis: SMT provides ∼1.3× speedup over single-threaded. With more physical cores, near-linear scaling is expected up to memory bandwidth limits.

Conclusion: Phase 2 achieves 15.4× total speedup on this 2-vCPU system. On a machine with 8+ cores, we expect 50+× speedup (target: 2 hours).

10.3 Phase 3: GPU Acceleration

Skipped—no GPU available on target system.

11 Conclusion

11.1 Summary of Optimizations

Phase	Optimization	Throughput	Speedup
0	Original hutter.c	2,800 bytes/s	1.0×
1	SIMD (AVX2 + FMA)	37,400 bytes/s	13.3×
2	+ OpenMP (2 threads)	42,700 bytes/s	15.2×

11.2 Time for enwik9 Forward Pass

Version	Time
Original	99 hours (4.1 days)
Phase 1 (SIMD)	7.4 hours
Phase 2 (+ OpenMP)	6.5 hours

11.3 Key Techniques

1. **Compiler flags**: `-O3 -march=native -ffast-math` provides 4× baseline improvement
2. **8-way loop unrolling**: Maximizes instruction-level parallelism
3. **FMA instructions**: Fused multiply-add in single cycle
4. **Fast tanh**: Padé polynomial approximation avoids expensive `tanhf`
5. **OpenMP parallelization**: Scales with available cores

11.4 Files

- `rnn_fast.h`: Drop-in SIMD replacement for `rnn_step()`
- `parallel_batch.c`: OpenMP parallel benchmark

11.5 Impact on Tick-Tock Cycle

With 15× speedup, a single forward pass takes 6.5 hours instead of 99 hours. Training epochs (multiple forward + backward passes) are now feasible in a day rather than a week. This enables practical iteration on the tick-tock methodology.