xLSTM: Extended Long Short-Term Memory

Original Paper: https://arxiv.org/pdf/2405.04517 [1]

Anubhab Biswas

anubhab.biswas@usi.ch
PhD Student, Faculty of Informatics
Università della Svizzera Italiana(USI)

January 17, 2025

Main motivation

- With the advent of the transformer models and parallelizable self-attention mechanisms, these models have outpaced LSTMs at scale.
- This raises a simple question - how far can we go with language modelling such that we can do the scaling of LSTM to billions of parameters using the latest LLM techniques but also mitigate the known limitations of LSTM?

Main contribution

- The authors introduce an exponential gating with appropriate normalization and stabilization techniques.
- They further modify the LSTM memory structure thereby obtaining - (i) sLSTM with a scalar memory, a scalar update, and new memory mixing, (ii) mLSTM that is fully parallelizable with a matrix memory and a covariance update rule.
- These extensions are integrated into residual block backbones giving rise to the xLSTM blocks that are then stacked residually into xLSTM architectures.
- Exponential gating and modified memory structures boost xLSTM capabilities to perform favorably when compared to state-of-the-art Transformers and State Space Models, both in performance and scaling.

LSTM model

- The Long-Short Term Memory (LSTM) model was introduced to overcome the vanishing gradient that occurs in Recurrent Neural Networks.

$$c_t = \mathrm{f}_t \; c_{t-1} + \mathrm{i}_t \; z_t \qquad \text{cell state} \quad (2)$$

$$h_t = \mathrm{o}_t \; \tilde{h}_t \,, \qquad \tilde{h}_t = \psi\left(c_t\right) \qquad \text{hidden state} \quad (3)$$

$$z_t = \varphi\left(\tilde{z}_t\right) \,, \qquad \tilde{z}_t = \boldsymbol{w}_z^\top \boldsymbol{x}_t + r_z h_{t-1} + b_z \qquad \text{cell input} \quad (4)$$

$$\mathrm{i}_t = \sigma\left(\tilde{\mathrm{i}}_t\right) \,, \qquad \tilde{\mathrm{i}}_t = \boldsymbol{w}_\mathrm{i}^\top \boldsymbol{x}_t + r_\mathrm{i} \, h_{t-1} + b_\mathrm{i} \qquad \text{input gate} \quad (5)$$

$$\mathrm{f}_t = \sigma\left(\tilde{\mathrm{f}}_t\right) \,, \qquad \tilde{\mathrm{f}}_t = \boldsymbol{w}_\mathrm{f}^\top \boldsymbol{x}_t + r_\mathrm{f} \, h_{t-1} + b_\mathrm{f} \qquad \text{forget gate} \quad (6)$$

$$\mathrm{o}_t = \sigma\left(\tilde{\mathrm{o}}_t\right) \,, \qquad \tilde{\mathrm{o}}_t = \boldsymbol{w}_\mathrm{o}^\top \boldsymbol{x}_t + r_\mathrm{o} \, h_{t-1} + b_\mathrm{o} \qquad \text{output gate} \quad (7)$$

The weight vectors $\boldsymbol{w}_z$, $\boldsymbol{w}_\mathrm{i}$, $\boldsymbol{w}_\mathrm{f}$, and $\boldsymbol{w}_\mathrm{o}$ correspond to the input weight vectors between inputs $\boldsymbol{x}_t$ and cell input, input gate, forget gate, and output gate, respectively. The weights r_z, r_i, r_f, and r_o correspond to the recurrent weights between hidden state h_{t-1} and cell input, input gate, forget gate, and output gate, respectively. b_z, b_i, b_f, and b_o are the corresponding bias terms. φ and ψ are the cell input and hidden state activation functions (typically $\tanh$). ψ is used to normalize or squash the cell state, which would be unbounded otherwise. All gate activation functions are sigmoid, i.e., $\sigma(x) = 1/(1+\exp(-x))$. In later formulations, multiple scalar memory cells $\boldsymbol{c}_t \in \mathbb{R}^d$ were combined in a vector, which allows the usage of recurrent weight matrices $\boldsymbol{R} \in \mathbb{R}^{d \times d}$ to mix the cell outputs of memory cells (Greff et al., 2015), for more details see Appendix A.1. Ablation studies showed that all components of the memory cell are crucial (Greff et al., 2015).

Main Limitations of LSTM

- **Inability to revise storage decisions**: LSTMs struggle to update previously stored information when better alternatives arise. For instance, when scanning a sequence to find the vector most similar to a given reference, LSTMs cant easily overwrite their initial choice if a more similar vector is encountered later. This is demonstrated in the Nearest Neighbor Search problem, where LSTMs exhibit higher error rates because they cannot revise their decisions.
- **Limited Storage Capacity**: The LSTM cell state is restricted to scalar values, which means information must be compressed, leading to poorer performance on tasks that depend on rare or infrequent data. In a Rare Token Prediction task (e.g., on Wikitext-103), LSTMs show higher perplexity on rare tokens because they lack sufficient capacity to handle them effectively.
- **Lack of Parallelizability**: The sequential dependency in LSTMs due to the way hidden states are mixed from one timestep to the next prevents parallel processing. This inherent sequential nature limits LSTMs ability to process data more efficiently.

Model Architecture

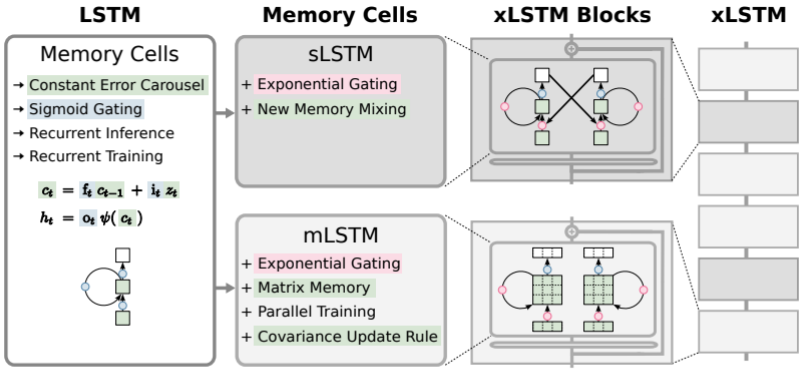


Figure 1: The extended LSTM (xLSTM) family. From left to right: 1. The original LSTM memory cell with constant error carousel and gating. 2. New sLSTM and mLSTM memory cells that introduce exponential gating. sLSTM offers a new memory mixing technique. mLSTM is fully parallelizable with a novel matrix memory cell state and new covariance update rule. 3. mLSTM and sLSTM in residual blocks yield xLSTM blocks. 4. Stacked xLSTM blocks give an xLSTM architecture.

Main Idea

- xLSTM addresses LSTM limitations by introducing:
- (i) **sLSTM**: Uses scalar memory with exponential gating, enabling enhanced storage and processing capabilities, and supports multiple cells and heads, mixing memory across cells within each head, but not across heads
- (ii) **mLSTM**: Employs matrix memory with a covariance update, fully parallelizable by removing hidden-hidden connections. Allows multiple cells and heads, treating them equivalently for flexible memory processing.
- By integrating sLSTM and mLSTM into residual blocks, xLSTM forms modular building blocks. Stacking these blocks produces advanced xLSTM architectures, offering improved scalability and performance.

sLSTM

- Input and forget gates can have exponential activation function.
- For normalization, a normalizer state that sums up the product of the input gate times all future forget gate is introduced.

$$c_t = \mathrm{f}_t \; c_{t-1} + \mathrm{i}_t \; z_t \qquad \text{cell state} \qquad (8)$$

$$n_t = \mathrm{f}_t \; n_{t-1} + \mathrm{i}_t \qquad \text{normalizer state} \qquad (9)$$

$$h_t = \mathrm{o}_t \; \tilde{h}_t \,, \qquad \tilde{h}_t = c_t \,/\, n_t \qquad \text{hidden state} \qquad (10)$$

$$z_t = \varphi\left(\tilde{z}_t\right) \,, \qquad \tilde{z}_t = \boldsymbol{w}_z^\top \boldsymbol{x}_t + r_z h_{t-1} + b_z \qquad \text{cell input} \qquad (11)$$

$$\mathrm{i}_t = \exp\left(\tilde{\mathrm{i}}_t\right) \,, \qquad \tilde{\mathrm{i}}_t = \boldsymbol{w}_\mathrm{i}^\top \boldsymbol{x}_t + r_\mathrm{i} \, h_{t-1} + b_\mathrm{i} \qquad \text{input gate} \qquad (12)$$

$$\mathrm{f}_t = \sigma\left(\tilde{\mathrm{f}}_t\right) \text{ OR } \exp\left(\tilde{\mathrm{f}}_t\right) \,, \qquad \tilde{\mathrm{f}}_t = \boldsymbol{w}_\mathrm{f}^\top \boldsymbol{x}_t + r_\mathrm{f} \, h_{t-1} + b_\mathrm{f} \qquad \text{forget gate} \qquad (13)$$

$$\mathrm{o}_t = \sigma\left(\tilde{\mathrm{o}}_t\right) \,, \qquad \tilde{\mathrm{o}}_t = \boldsymbol{w}_\mathrm{o}^\top \boldsymbol{x}_t + r_\mathrm{o} \, h_{t-1} + b_\mathrm{o} \qquad \text{output gate} \qquad (14)$$

sLSTM

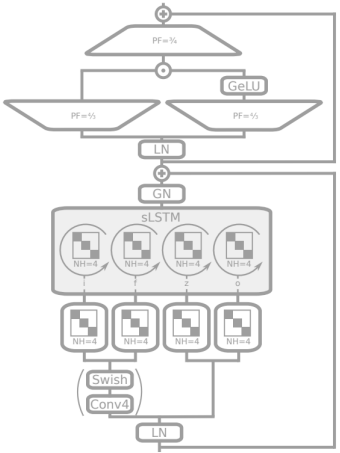
- Exponential activation functions can lead to large values that cause overflows. Therefore, the gates are stabilized with an additional state m_t.

$$m_t = \max\left(\log(\mathrm{f}_t) + m_{t-1}, \log(\mathrm{i}_t)\right) \quad \text{stabilizer state} \quad (15)$$

$$\mathrm{i}'_t = \exp\left(\log\left(\mathrm{i}_t\right) - m_t\right) = \exp\left(\tilde{\mathrm{i}}_t - m_t\right) \quad \text{stabil. input gate} \quad (16)$$

$$\mathrm{f}'_t = \exp\left(\log\left(\mathrm{f}_t\right) + m_{t-1} - m_t\right) \quad \text{stabil. forget gate} \quad (17)$$

sLSTM Architecture



mLSTM

- To enhance the storage capacities of LSTMs, the mLSTM replaces the scalar memory cell $c \in \mathbb{R}$ with a **matrix memory** $C \in \mathbb{R}^{d \times d}$. The retrieval process is performed via a matrix multiplication. At time t, given a key vector $k_t \in \mathbb{R}^d$ and a value vector $v_t \in \mathbb{R}^d$, the memory is updated using the **covariance update rule**:

$$C_t = C_{t-1} + v_t k_t^\top$$

- Inputs to keys and values undergo layer normalization to ensure zero mean and unit variance, improving stability and separability during retrieval.
- The covariance update ensures maximal separability of retrieved vectors, equivalent to maximizing signal-to-noise ratio. This enables efficient pairwise interactions while limiting quadratic complexity.
- The forget gate corresponds to the decay rate of the memory matrix, while the input gate corresponds to the learning rate. The output gate controls how the retrieved vector is scaled and used.
- A weighted sum of key vectors is maintained, with weights determined by input and forget gates. This state tracks the strength of gates and ensures robust retrieval.
- The dot product between query and normalizer states is constrained (e.g., lower-bounded by a threshold like 1.0) to maintain numerical stability.

mLSTM

$$\boldsymbol{C}_t = \mathrm{f}_t \; \boldsymbol{C}_{t-1} + \mathrm{i}_t \; \boldsymbol{v}_t \, \boldsymbol{k}_t^\top \qquad \text{cell state} \quad (19)$$

$$\boldsymbol{n}_t = \mathrm{f}_t \; \boldsymbol{n}_{t-1} + \mathrm{i}_t \; \boldsymbol{k}_t \qquad \text{normalizer state} \quad (20)$$

$$\boldsymbol{h}_t = \mathbf{o}_t \odot \tilde{\boldsymbol{h}}_t \,, \qquad \tilde{\boldsymbol{h}}_t = \boldsymbol{C}_t \; \boldsymbol{q}_t \; / \; \max\left\{ \left| \boldsymbol{n}_t^\top \; \boldsymbol{q}_t \right|, 1 \right\} \qquad \text{hidden state} \quad (21)$$

$$\boldsymbol{q}_t = \boldsymbol{W}_q \, \boldsymbol{x}_t + \boldsymbol{b}_q \qquad \text{query input} \quad (22)$$

$$\boldsymbol{k}_t = \frac{1}{\sqrt{d}} \boldsymbol{W}_k \, \boldsymbol{x}_t + \boldsymbol{b}_k \qquad \text{key input} \quad (23)$$

$$\boldsymbol{v}_t = \boldsymbol{W}_v \, \boldsymbol{x}_t + \boldsymbol{b}_v \qquad \text{value input} \quad (24)$$

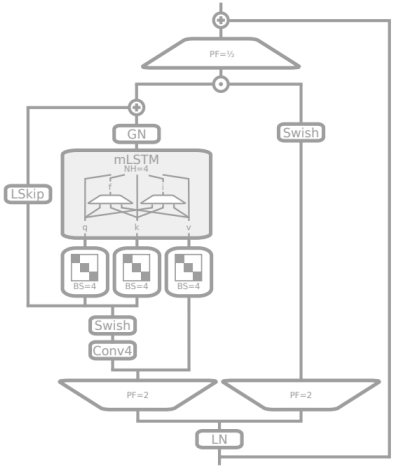
$$\mathrm{i}_t = \exp\left(\tilde{\mathrm{i}}_t\right) \,, \qquad \tilde{\mathrm{i}}_t = \boldsymbol{w}_\mathrm{i}^\top \, \boldsymbol{x}_t + b_\mathrm{i} \qquad \text{input gate} \quad (25)$$

$$\mathrm{f}_t = \sigma\left(\tilde{\mathrm{f}}_t\right) \text{ OR } \exp\left(\tilde{\mathrm{f}}_t\right), \quad \tilde{\mathrm{f}}_t = \boldsymbol{w}_\mathrm{f}^\top \, \boldsymbol{x}_t + b_\mathrm{f} \qquad \text{forget gate} \quad (26)$$

$$\mathbf{o}_t = \sigma\left(\tilde{\mathbf{o}}_t\right) \,, \qquad \tilde{\mathbf{o}}_t = \boldsymbol{W}_\mathbf{o} \, \boldsymbol{x}_t + \boldsymbol{b}_\mathbf{o} \qquad \text{output gate} \quad (27)$$

- In order to stabilize the exponential gates of mLSTM, the same stabilization techniques as for sLST are used.
- Since the mLSTM has no memory mixing, this recurrence can be reformulated in a parallel version.

mLSTM architecture



xLSTM

- The xLSTM block is designed to non-linearly summarize past sequence information in a high-dimensional space to better separate different histories or contexts.
- This separation is crucial for accurately predicting the next sequence element, such as the next token.
- The approach is inspired by Cover's Theorem, which states that patterns are more likely to be linearly separable when embedded in a higher-dimensional space.
- The xLSTM framework uses two types of residual block architectures for this purpose.

xLSTM

- (i) **Post Up-Projection Block** (similar to Transformers):
- The past is first non-linearly summarized in the *original space*. It is then linearly mapped into a *high-dimensional space*, followed by a non-linear activation. Finally, it is linearly mapped back to the *original space*.
- This architecture is used for xLSTM blocks containing sLSTM, as the memory operations are suited for the original space.
- (ii) **Pre Up-Projection Block (similar to State Space Models)**:
- The input is first linearly mapped into a high-dimensional space. The past is then non-linearly summarized within the high-dimensional space. Finally, it is linearly mapped back to the original space.
- This architecture is used for xLSTM blocks containing mLSTM, as the larger memory capacity benefits from the high-dimensional space.

Memory and Speed Considerations

- Unlike Transformers, xLSTM networks have linear computation and constant memory complexity with respect to sequence length, making them efficient for industrial applications and edge devices.
- The mLSTM uses a $d \times d$ matrix memory and updates, which increases computational cost. Despite higher computational complexity, mLSTM computations can be parallelized on GPUs, minimizing the impact on wall clock time.
- mLSTM is fully parallelizable, similar to techniques like FlashAttention or GLA, enabling efficient GPU acceleration. sLSTM, however, is not parallelizable due to its memory mixing (hidden-hidden connections).
- A fast CUDA implementation with GPU memory optimizations was developed for sLSTM, reducing its computational overhead. It is typically less than two times slower than mLSTM.

Test of xLSTMs Exponential Gating with Memory Mixing

	Context Sensitive		Deterministic Context Free		Regular					
	Bucket Sort	Missing Duplicate	Mod Arithmetic (w Brackets)	Solve Equation	Cycle Nav	Even Pairs	Mod Arithmetic (w/o Brackets)	Parity	Majority	Majority Count
Llama	0.92 ± 0.02	0.08 ± 0.0	0.02 ± 0.0	0.02 ± 0.0	0.04 ± 0.01	1.0 ± 0.0	0.03 ± 0.0	0.03 ± 0.01	0.37 ± 0.01	0.13 ± 0.0
Mamba	0.69 ± 0.0	0.15 ± 0.0	0.04 ± 0.01	0.05 ± 0.02	0.86 ± 0.04	1.0 ± 0.0	0.05 ± 0.02	0.13 ± 0.02	0.69 ± 0.01	0.45 ± 0.03
Retention	0.13 ± 0.01	0.03 ± 0.0	0.03 ± 0.0	0.03 ± 0.0	0.05 ± 0.01	0.51 ± 0.07	0.04 ± 0.0	0.05 ± 0.01	0.36 ± 0.0	0.12 ± 0.01
Hyena	0.3 ± 0.02	0.06 ± 0.02	0.05 ± 0.0	0.02 ± 0.0	0.06 ± 0.01	0.93 ± 0.07	0.04 ± 0.0	0.04 ± 0.0	0.36 ± 0.01	0.18 ± 0.02
RWKV-4	0.54 ± 0.0	0.21 ± 0.01	0.06 ± 0.0	0.07 ± 0.0	0.13 ± 0.0	1.0 ± 0.0	0.07 ± 0.0	0.06 ± 0.0	0.63 ± 0.0	0.13 ± 0.0
RWKV-5	0.49 ± 0.04	0.15 ± 0.01	0.08 ± 0.0	0.08 ± 0.0	0.26 ± 0.05	1.0 ± 0.0	0.15 ± 0.02	0.06 ± 0.03	0.73 ± 0.01	0.34 ± 0.03
RWKV-6	0.96 ± 0.0	0.23 ± 0.06	0.09 ± 0.01	0.09 ± 0.02	0.31 ± 0.14	1.0 ± 0.0	0.16 ± 0.0	0.22 ± 0.12	0.76 ± 0.01	0.24 ± 0.01
LSTM (Block)	0.99 ± 0.0	0.15 ± 0.0	0.76 ± 0.0	0.5 ± 0.05	0.97 ± 0.03	1.0 ± 0.0	0.91 ± 0.09	1.0 ± 0.0	0.58 ± 0.02	0.27 ± 0.0
LSTM	0.94 ± 0.01	0.2 ± 0.0	0.72 ± 0.04	0.38 ± 0.05	0.93 ± 0.07	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	0.82 ± 0.02	0.33 ± 0.0
xLSTM[0:1]	0.84 ± 0.08	0.23 ± 0.01	0.57 ± 0.09	0.55 ± 0.09	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	0.75 ± 0.02	0.22 ± 0.0
xLSTM[1:0]	0.97 ± 0.0	0.33 ± 0.22	0.03 ± 0.0	0.03 ± 0.01	0.86 ± 0.03	1.0 ± 0.0	0.04 ± 0.0	0.04 ± 0.01	0.74 ± 0.01	0.46 ± 0.0
xLSTM[1:1]	0.7 ± 0.23	0.2 ± 0.01	0.15 ± 0.06	0.24 ± 0.04	0.8 ± 0.03	1.0 ± 0.0	0.6 ± 0.4	1.0 ± 0.0	0.64 ± 0.04	0.5 ± 0.0

Figure 4: Test of xLSTM's exponential gating with memory mixing. Results are given by the scaled accuracy of different models at solving formal language tasks, of which some require state tracking. The different tasks are grouped by the Chomsky hierarchy.

xLSTMs Memory Capacities on Associative Recall Tasks

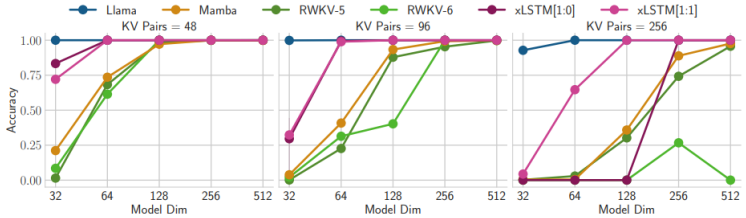


Figure 5: Test of memory capacities of different models at the Multi-Query Associative Recall task with context length 2048. Each panel is dedicated to a different number of key-value pairs. The x-axis displays the model size and the y-axis the validation accuracy.

- xLSTM demonstrates superior performance in tasks requiring state tracking, thanks to its memory mixing and exponential gating, compared to models that lack these features.
- While comparing Memory Capacities on Associative Recall Tasks xLSTM demonstrates superior memory capacity among non-Transformer models.

Limitations

- The memory mixing in sLSTM prevents parallelizable operations, making it slower than mLSTM. Despite this, a fast CUDA kernel for sLSTM has been developed, but it remains less than two times slower than the parallel mLSTM implementation.
- The current mLSTM CUDA kernels are not fully optimized, making it 4 times slower than FlashAttention or the scan used in Mamba. Optimized kernels could significantly improve performance.
- The $d \times d$ matrix memory in mLSTM increases computational complexity. However, memory updates and retrieval are parameter-free and parallelizable with standard matrix operations, minimizing wall clock time overhead.
- The forget gate initialization needs to be carefully chosen to ensure stable performance.
- Although the matrix memory is independent of sequence length, longer contexts might overload the memory.
- Due to computational costs, the architecture and hyperparameters of xLSTM have not been fully optimized, especially for larger models. Further optimization is required to unlock the full potential of xLSTM.

Conclusion

- xLSTM performs competitively with state-of-the-art models like Transformers and State Space Models in language modeling tasks.
- xLSTM leverages exponential gating with memory mixing and a new memory structure, enabling improved performance and scalability.
- Scaling laws suggest that larger xLSTM models could become strong competitors to Transformer-based Large Language Models.
- Beyond language modeling, xLSTM has the potential to significantly impact fields like Reinforcement Learning, Time Series Prediction, and Physical Systems Modeling.

References I

[1] Maximilian Beck et al. *xLSTM: Extended Long Short-Term Memory*. 2024. arXiv: 2405.04517 [cs.LG]. URL: https://arxiv.org/abs/2405.04517.

Thank you for listening !

Anubhab Biswas

@anubhab.biswas@usi.ch
@anubhab.biswas@supsi.ch

xLSTM as a Large Language Model (LLM)

- xLSTM excels at sequence length extrapolation, maintaining high performance on extremely long contexts.
- It consistently achieves state-of-the-art results in validation perplexity, downstream tasks, and language modeling benchmarks, making it a robust and versatile LLM.
- The xLSTM[1:0] variant is particularly effective, especially on fine-grained domain-specific tasks.