xLSTM

Nadi Tomeh
Groupe de lecture RCLN

November 4, 2024

Motivation for Exploring xLSTM I

Large-Scale Language Modeling

- Emergence of advanced language skills at scale.
- **Transformers**: large number of parameters trained in parallel.

Open Question

- Does the choice of architecture matter, or will any model scale effectively?
- **Candidate Architectures** include State Space Models (SSMs) and Recurrent Models.

Motivation for Exploring xLSTM II

Theoretical Results

- Transformers and SSMs are in TC^0 (Merril et al. TACL'23; ICML'24)
  Problems solvable by constant-depth, polynomial-size circuits composed of AND, OR, NOT, and threshold gates.
- Sequential problems like permutation computation (S_5) are outside TC^0 but can be solved by RNNs (Minksy 54). Simulating FSA is in NC^1-complete, graph connectivity (L-complete), solving linear equations (P-complete), etc.

Empirical Results

- **RNNs and Transformers** don't generalize on *non-regular* tasks; **LSTMs** can solve *regular* and *counter-language* tasks; only networks augmented with **structured memory** can generalize on *context-free* and *context-sensitive* tasks (Delétang et al. ICLR'23).

Outline

RNN

LSTM

xLSTM

Experiments

RNN Architecture Overview

- RNNs maintain a hidden state that updates over time.
- Hidden state $\boldsymbol{h}_t$ captures information from previous time steps.
- Update equation:

$$\boldsymbol{h}_t = \tanh(\boldsymbol{W}_{hh}\boldsymbol{h}_{t-1} + \boldsymbol{W}_{xh}\boldsymbol{x}_t + \boldsymbol{b}_h)$$

- Output equation:

$$\boldsymbol{y}_t = \boldsymbol{W}_{hy}\boldsymbol{h}_t + \boldsymbol{b}_y$$

- Key components:
  - Input vector $\boldsymbol{x}_t \in \mathbb{R}^n$
  - Hidden state $\boldsymbol{h}_t \in \mathbb{R}^d$
  - Output vector $\boldsymbol{y}_t \in \mathbb{R}^m$
  - Weight matrices:

$$\boldsymbol{W}_{hh} \in \mathbb{R}^{d\times d}, \quad \boldsymbol{W}_{xh} \in \mathbb{R}^{d\times n}, \quad \boldsymbol{W}_{hy} \in \mathbb{R}^{m\times d}$$

  - Bias terms: $\boldsymbol{b}_h \in \mathbb{R}^d$, $\boldsymbol{b}_y \in \mathbb{R}^m$

RNN Forward Pass Example with Sequence Length 4

- Consider a sequence $\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3, \boldsymbol{x}_4$, where $\boldsymbol{x}_t \in \mathbb{R}^n$.
- Compute hidden state at time step 4 ($\boldsymbol{h}_4$):

$$\begin{aligned}\boldsymbol{h}_4 = \tanh\Big(&\boldsymbol{W}_{hh} \tanh\Big(\\ &\quad \boldsymbol{W}_{hh} \tanh\Big(\\ &\quad\quad \boldsymbol{W}_{hh} \tanh\Big(\\ &\quad\quad\quad \boldsymbol{W}_{hh} \boldsymbol{h}_0 + \boldsymbol{W}_{xh}\boldsymbol{x}_1 + \boldsymbol{b}_h\Big)\\ &\quad\quad + \boldsymbol{W}_{xh}\boldsymbol{x}_2 + \boldsymbol{b}_h\Big)\\ &\quad + \boldsymbol{W}_{xh}\boldsymbol{x}_3 + \boldsymbol{b}_h\Big)\\ &+ \boldsymbol{W}_{xh}\boldsymbol{x}_4 + \boldsymbol{b}_h\Big)\end{aligned}$$

- Output at time step 4:

$$\boldsymbol{y}_4 = \boldsymbol{W}_{hy}\boldsymbol{h}_4 + \boldsymbol{b}_y$$

- $\boldsymbol{h}_4$ depends on $\boldsymbol{h}_0$ and all previous inputs $\boldsymbol{x}_1$ to $\boldsymbol{x}_4$, with each input influencing the hidden state through multiple applications of tanh.

RNN Training Challenges

- Training involves computing gradients through time, known as **Backpropagation Through Time (BPTT)**.
- RNNs often suffer from **vanishing** or **exploding** gradients, making training difficult.

Loss Function Over Time Steps

- The total loss L is the sum over all time steps:

$$L = \sum_{t=1}^{T} L_t(\boldsymbol{y}_t, \hat{\boldsymbol{y}}_t)$$

- $\boldsymbol{y}_t$: True output at time t.
- $\hat{\boldsymbol{y}}_t$: Predicted output at time t.
- L_t: Loss at time t (e.g., cross-entropy or MSE).

Gradient w.r.t. Hidden State

- Goal: Compute $\frac{\partial L}{\partial \boldsymbol{h}_t}$.
- Using the chain rule:

$$\frac{\partial L}{\partial \boldsymbol{h}_t} = \frac{\partial L_t}{\partial \boldsymbol{h}_t} + \sum_{k=t+1}^{T} \frac{\partial L_k}{\partial \boldsymbol{h}_t}$$

- Simplifies to a recursive formula:

$$\frac{\partial L}{\partial \boldsymbol{h}_t} = \frac{\partial L_t}{\partial \boldsymbol{h}_t} + \frac{\partial L}{\partial \boldsymbol{h}_{t+1}} \frac{\partial \boldsymbol{h}_{t+1}}{\partial \boldsymbol{h}_t}$$

- This accounts for both direct and indirect dependencies.

Computing the Recursive Gradient I

- Recall the hidden state update:

$$\boldsymbol{h}_{t+1} = \tanh(\boldsymbol{a}_{t+1})$$

where

$$\boldsymbol{a}_{t+1} = \boldsymbol{W}_{hh}\boldsymbol{h}_t + \boldsymbol{W}_{xh}\boldsymbol{x}_{t+1} + \boldsymbol{b}_h$$

- Compute the derivative of $\boldsymbol{h}_{t+1}$ with respect to $\boldsymbol{h}_t$:

$$\frac{\partial \boldsymbol{h}_{t+1}}{\partial \boldsymbol{h}_t} = \frac{\partial \boldsymbol{h}_{t+1}}{\partial \boldsymbol{a}_{t+1}} \frac{\partial \boldsymbol{a}_{t+1}}{\partial \boldsymbol{h}_t}$$

- Compute $\frac{\partial \boldsymbol{a}_{t+1}}{\partial \boldsymbol{h}_t}$:

$$\frac{\partial \boldsymbol{a}_{t+1}}{\partial \boldsymbol{h}_t} = \boldsymbol{W}_{hh}$$

$\boldsymbol{W}_{xh}\boldsymbol{x}_{t+1}$ and $\boldsymbol{b}_h$ are constants with respect to $\boldsymbol{h}_t$.

Computing the Recursive Gradient II

- Compute $\frac{\partial \boldsymbol{h}_{t+1}}{\partial \boldsymbol{a}_{t+1}}$:

$$\frac{\partial \boldsymbol{h}_{t+1}}{\partial \boldsymbol{a}_{t+1}} = \operatorname{diag}\left(1 - \tanh^2(\boldsymbol{a}_{t+1})\right)$$

  - Since tanh is applied element-wise, its derivative is a diagonal matrix.
  - Each diagonal element corresponds to $1 - \tanh^2(a_{t+1}^{(i)})$.
- Combine the derivatives:

$$\frac{\partial \boldsymbol{h}_{t+1}}{\partial \boldsymbol{h}_t} = \operatorname{diag}\left(1 - \tanh^2(\boldsymbol{a}_{t+1})\right) \boldsymbol{W}_{hh}$$

Computing the Recursive Gradient III

- Update the gradient expression:

$$\frac{\partial L}{\partial \boldsymbol{h}_t} = \frac{\partial L_t}{\partial \boldsymbol{h}_t} + \left(\frac{\partial L}{\partial \boldsymbol{h}_{t+1}} \frac{\partial \boldsymbol{h}_{t+1}}{\partial \boldsymbol{h}_t} \right)$$

  Substitute $\frac{\partial \boldsymbol{h}_{t+1}}{\partial \boldsymbol{h}_t}$ into the equation:

$$\frac{\partial L}{\partial \boldsymbol{h}_t} = \frac{\partial L_t}{\partial \boldsymbol{h}_t} + \left(\frac{\partial L}{\partial \boldsymbol{h}_{t+1}} \mathrm{diag}\left(1 - \tanh^2(\boldsymbol{a}_{t+1})\right) \boldsymbol{W}_{hh} \right)$$

- Adjust for correct dimensions (Transpose):

$$\boxed{\frac{\partial L}{\partial \boldsymbol{h}_t} = \frac{\partial L_t}{\partial \boldsymbol{h}_t} + \boldsymbol{W}_{hh}^\top \left(\mathrm{diag}\left(1 - \tanh^2(\boldsymbol{a}_{t+1})\right) \frac{\partial L}{\partial \boldsymbol{h}_{t+1}} \right)}$$

Unrolling the Recursion

- Apply the recursive formula repeatedly:

$$
\begin{aligned}
\frac{\partial L}{\partial \boldsymbol{h}_t} &= \frac{\partial L_t}{\partial \boldsymbol{h}_t} + \boldsymbol{W}_{hh}^{\top} \boldsymbol{\Phi}'_{t+1} \frac{\partial L}{\partial \boldsymbol{h}_{t+1}} \\
&= \frac{\partial L_t}{\partial \boldsymbol{h}_t} + \boldsymbol{W}_{hh}^{\top} \boldsymbol{\Phi}'_{t+1} \left(\frac{\partial L_{t+1}}{\partial \boldsymbol{h}_{t+1}} + \boldsymbol{W}_{hh}^{\top} \boldsymbol{\Phi}'_{t+2} \frac{\partial L}{\partial \boldsymbol{h}_{t+2}} \right) \\
&\vdots \\
&= \sum_{k=t}^{T} \left(\left(\prod_{j=t+1}^{k} \boldsymbol{W}_{hh}^{\top} \boldsymbol{\Phi}'_j \right) \frac{\partial L_k}{\partial \boldsymbol{h}_k} \right)
\end{aligned}
$$

- The product operator $\prod$ represents matrix multiplication over time steps.
- It highlights the accumulation of gradients over time.
- The product term influences the magnitude of the gradients.

Vanishing Gradients

- When $\|\boldsymbol{W}_{hh}\|_2 < 1$:
  - The product $\prod_{j=t+1}^{k} \boldsymbol{W}_{hh}^{\top} \boldsymbol{\Phi}_j'$ decreases exponentially.
  - Gradients $\frac{\partial L}{\partial \boldsymbol{h}_t}$ become very small.
- Result: Difficulty in learning long-term dependencies.
- Illustration:

$$\left\| \frac{\partial L}{\partial \boldsymbol{h}_t} \right\| \leq (\|\boldsymbol{W}_{hh}\|_2 \cdot \gamma)^{(k-t)} \left\| \frac{\partial L_k}{\partial \boldsymbol{h}_k} \right\|$$

- Where $\gamma = \max_j \|\boldsymbol{\Phi}_j'\|_2 \leq 1$.

Exploding Gradients

- When $\|\boldsymbol{W}_{hh}\|_2 > 1$:
  - The product $\prod_{j=t+1}^{k} \boldsymbol{W}_{hh}^{\top} \boldsymbol{\Phi}_j'$ increases exponentially.
  - Gradients $\frac{\partial L}{\partial \boldsymbol{h}_t}$ become very large.
- Result: Numerical instability during training.
- Illustration:

$$\left\| \frac{\partial L}{\partial \boldsymbol{h}_t} \right\| \geq \left(\|\boldsymbol{W}_{hh}\|_2 \cdot \gamma\right)^{(k-t)} \left\| \frac{\partial L_k}{\partial \boldsymbol{h}_k} \right\|$$

Theoretical Understanding

- Spectral Radius $\rho(\boldsymbol{W}_{hh})$:
  - Largest absolute eigenvalue of $\boldsymbol{W}_{hh}$.
  - $\rho(\boldsymbol{W}_{hh}) < 1$ leads to vanishing gradients.
  - $\rho(\boldsymbol{W}_{hh}) > 1$ leads to exploding gradients.
- Lyapunov Exponents:
  - Measure divergence/convergence rates in dynamical systems.
  - Negative exponents: Vanishing gradients.
  - Positive exponents: Exploding gradients.
- Norms of Jacobians:
  - Norms $\|\frac{\partial \boldsymbol{h}_{t+1}}{\partial \boldsymbol{h}_t}\|$ affect gradient magnitude.

Mitigating Gradient Problems

- **Gradient Clipping**:
  - Restricts the gradient norm to a predefined threshold.
- **Initialization Techniques**:
  - Properly initializing weights to maintain stable gradients.
  - Use of orthogonal matrices for $\boldsymbol{W}_{hh}$: $\boldsymbol{W}_{hh}\boldsymbol{W}_{hh}^{\top} = \boldsymbol{I}$. Preserves the norm of vectors during multiplication: $\|\boldsymbol{W}_{hh}\boldsymbol{x}\| = \|\boldsymbol{x}\|$
- **Activation Functions**:
  - Use ReLU variants: $\mathrm{ReLU}(x) = \max(0, x)$. Derivative is 1 for positive inputs, allowing gradients to flow back without shrinking.
- **Advanced RNN Architectures**:
  - **Long Short-Term Memory (LSTM)** Hochreiter and Schmidhuber (1997) introduce constant error carrousel and gates to control information flow.
  - **Gated Recurrent Units (GRU)** simplify LSTMs while addressing gradient issues.

LSTM Architecture: Scalar and Vector Forms I

- LSTM memory cell update rules at time step t:
  Scalar Form:

$$
\begin{aligned}
c_t &= f_t \cdot c_{t-1} + i_t \cdot z_t && \text{cell state} \\
h_t &= o_t \cdot \tilde{h}, \quad \tilde{h} = \psi(c_t) && \text{hidden state} \\
z_t &= \varphi(\tilde{z}_t), \quad \tilde{z}_t = \boldsymbol{w}_z^\top \boldsymbol{x}_t + r_z h_{t-1} + b_z && \text{cell input} \\
i_t &= \sigma(\tilde{i}_t), \quad \tilde{i}_t = \boldsymbol{w}_i^\top \boldsymbol{x}_t + r_i h_{t-1} + b_i && \text{input gate} \\
f_t &= \sigma(\tilde{f}_t), \quad \tilde{f}_t = \boldsymbol{w}_f^\top \boldsymbol{x}_t + r_f h_{t-1} + b_f && \text{forget gate} \\
o_t &= \sigma(\tilde{o}_t), \quad \tilde{o}_t = \boldsymbol{w}_o^\top \boldsymbol{x}_t + r_o h_{t-1} + b_o && \text{output gate}
\end{aligned}
$$

- Here, $\boldsymbol{x}_t \in \mathbb{R}^n$ and $\boldsymbol{w}_z, \boldsymbol{w}_i, \boldsymbol{w}_f, \boldsymbol{w}_o \in \mathbb{R}^n$ are input weight vectors, while $r_z, r_i, r_f, r_o \in \mathbb{R}$ are scalar recurrent weights.

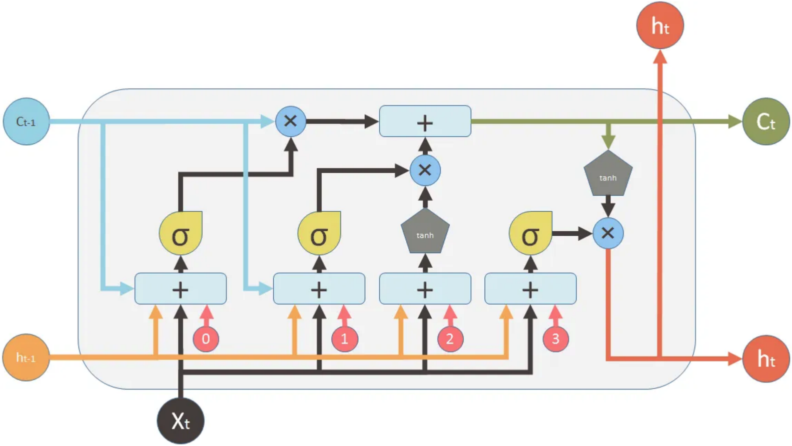
LSTM Architecture: Scalar and Vector Forms II

- **Vector Form:**

$$\begin{aligned}
\boldsymbol{c}_t &= \boldsymbol{f}_t \odot \boldsymbol{c}_{t-1} + \boldsymbol{i}_t \odot \boldsymbol{z}_t && \text{cell state } (\in \mathbb{R}^d) \\
\boldsymbol{h}_t &= \boldsymbol{o}_t \odot \psi(\boldsymbol{c}_t) && \text{hidden state } (\in \mathbb{R}^d) \\
\boldsymbol{z}_t &= \varphi(\tilde{\boldsymbol{z}}_t), \quad \tilde{\boldsymbol{z}}_t = \boldsymbol{W}_z \boldsymbol{x}_t + \boldsymbol{R}_z \boldsymbol{h}_{t-1} + \boldsymbol{b}_z && \text{cell input} \\
\boldsymbol{i}_t &= \sigma(\tilde{\boldsymbol{i}}_t), \quad \tilde{\boldsymbol{i}}_t = \boldsymbol{W}_i \boldsymbol{x}_t + \boldsymbol{R}_i \boldsymbol{h}_{t-1} + \boldsymbol{b}_i && \text{input gate} \\
\boldsymbol{f}_t &= \sigma(\tilde{\boldsymbol{f}}_t), \quad \tilde{\boldsymbol{f}}_t = \boldsymbol{W}_f \boldsymbol{x}_t + \boldsymbol{R}_f \boldsymbol{h}_{t-1} + \boldsymbol{b}_f && \text{forget gate} \\
\boldsymbol{o}_t &= \sigma(\tilde{\boldsymbol{o}}_t), \quad \tilde{\boldsymbol{o}}_t = \boldsymbol{W}_o \boldsymbol{x}_t + \boldsymbol{R}_o \boldsymbol{h}_{t-1} + \boldsymbol{b}_o && \text{output gate}
\end{aligned}$$

- Multiple memory cells are combined into a vector representation ($\boldsymbol{c}_t$ and $\boldsymbol{h}_t$)
- Allows the use of recurrent weight matrices ($\boldsymbol{R}_z, \boldsymbol{R}_i, \boldsymbol{R}_f, \boldsymbol{R}_o$) to **mix** the outputs of memory cells.
- Crucial for capturing complex dependencies across time steps (Greff et al. 2015).

LSTM Architecture: information flow



https://blog.mlreview.com/understanding-lstm-and-its-diagrams-37e2f46f1714

How the Constant Error Carousel Solves Vanishing Gradients I

- **Constant Error Carousel (CEC) in LSTM**:
  - Introduced by Hochreiter and Schmidhuber (1997); Gers et al. (2000) added the forget gate:

$$\boldsymbol{c}_t = \boldsymbol{f}_t \odot \boldsymbol{c}_{t-1} + \boldsymbol{i}_t \odot \boldsymbol{z}_t$$

  - Additive Updates:
    - Cell state $\boldsymbol{c}_t \in \mathbb{R}^{n_h}$ updated via element-wise operations.
    - Avoids multiplication that can shrink gradients.
- **Gradient Flow through CEC**:
  - Recursive Gradient Equation:

$$\frac{\partial L}{\partial \boldsymbol{c}_t} = \frac{\partial L_t}{\partial \boldsymbol{c}_t} + \left(\frac{\partial L}{\partial \boldsymbol{c}_{t+1}} \odot \boldsymbol{f}_{t+1} \right)$$

How the Constant Error Carousel Solves Vanishing Gradients II

- Unrolling the Recursion:

$$
\begin{aligned}
\frac{\partial L}{\partial \boldsymbol{c}_t} &= \frac{\partial L_t}{\partial \boldsymbol{c}_t} + \left(\left[\frac{\partial L_{t+1}}{\partial \boldsymbol{c}_{t+1}} + \left(\frac{\partial L}{\partial \boldsymbol{c}_{t+2}} \odot \boldsymbol{f}_{t+2} \right) \right] \odot \boldsymbol{f}_{t+1} \right) \\
&= \frac{\partial L_t}{\partial \boldsymbol{c}_t} + \left(\frac{\partial L_{t+1}}{\partial \boldsymbol{c}_{t+1}} \odot \boldsymbol{f}_{t+1} \right) + \left(\frac{\partial L}{\partial \boldsymbol{c}_{t+2}} \odot \boldsymbol{f}_{t+2} \odot \boldsymbol{f}_{t+1} \right) \\
&\vdots \\
&= \sum_{k=t}^{T} \left(\left(\frac{\partial L_k}{\partial \boldsymbol{c}_k} \odot \prod_{j=t+1}^{k} \boldsymbol{f}_j \right) \right)
\end{aligned}
$$

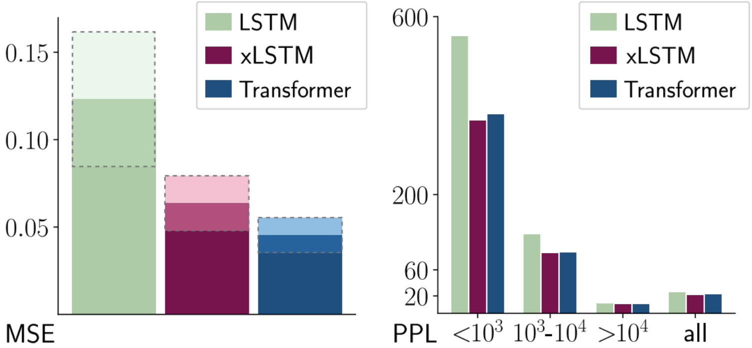
- The gradient $\frac{\partial L}{\partial \boldsymbol{c}_t}$ accumulates contributions from future time steps.
- Each term is modulated by the product of forget gates $\boldsymbol{f}_j$ with elements in $[0, 1]$, controlling gradient flow. If $\boldsymbol{f}_j$ elements are close to 1, gradients are preserved.

Do LSTMs Actually Have Long Memory?

- **Bengio et al. (1994)**: Showed difficulty in learning long-term dependencies with GD in systems like $y_t = M(y_{t-1}) + \varepsilon_t$.
- **Cheng et al. (2016)**: Pointed out that LSTM updates are Markovian and fit Bengio's system.
- **Miller & Hardt (2018)**: Proved that r-step LSTM is *stable*, which limits modeling of long-range dependencies.
- **Greaves-Tunnell & Harchaoui (2019)**:
  - Defined long dependency in terms of long memory in stochastic processes. Long memory in RNNs can be re-framed as a comparison between a learned representation and an estimated property of the data.
  - Langauge data has long dependencies, not captured by RNNs.
- **Zhao et al. (2020)**:
  - Represent RNN/LSTM as *markovian network processes*
  - Show short memory by showing that the process is *geometrically ergodic*, meaning that the dependency on initial states decays exponentially over time.

Limitations of LSTM Networks addressed by xLSTM

1. Inability to Revise Storage Decisions
   - *Example*: Nearest Neighbor Search task.
2. Limited Storage Capacities
   - *Example*: Poor performance on Rare Token Prediction.
3. Lack of Parallelizability Due to Memory Mixing
   - Hidden-to-hidden connections enforce sequential processing.
   - Limits efficient computation on modern hardware.



sLSTM Forward Pass and Stabilization I

sLSTM forward pass equations:

$$\boldsymbol{c}_t = \boldsymbol{f}_t \odot \boldsymbol{c}_{t-1} + \boldsymbol{i}_t \odot \boldsymbol{z}_t \qquad \text{cell state}$$

$$\boxed{\boldsymbol{n}_t} = \boldsymbol{f}_t \odot \boldsymbol{n}_{t-1} + \boldsymbol{i}_t \qquad \text{normalizer state}$$

$$\boldsymbol{h}_t = \boldsymbol{o}_t \odot \tilde{\boldsymbol{h}}, \quad \tilde{\boldsymbol{h}} = \boxed{\frac{\boldsymbol{c}_t}{\boldsymbol{n}_t}} \qquad \text{hidden state}$$

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

$$\boldsymbol{i}_t = \boxed{\exp}(\tilde{\boldsymbol{i}}_t), \quad \tilde{\boldsymbol{i}}_t = \boldsymbol{w}_i^\top \boldsymbol{x}_t + r_i \boldsymbol{h}_{t-1} + b_i \qquad \text{input gate}$$

$$\boldsymbol{f}_t = \sigma(\tilde{\boldsymbol{f}}_t) \text{ or } \boxed{\exp}(\tilde{\boldsymbol{f}}_t), \quad \tilde{\boldsymbol{f}}_t = \boldsymbol{w}_f^\top \boldsymbol{x}_t + r_f \boldsymbol{h}_{t-1} + b_f \qquad \text{forget gate}$$

$$\boldsymbol{o}_t = \sigma(\tilde{\boldsymbol{o}}_t), \quad \tilde{\boldsymbol{o}}_t = \boldsymbol{w}_o^\top \boldsymbol{x}_t + r_o \boldsymbol{h}_{t-1} + b_o \qquad \text{output gate}$$

- Cell input activation function is tanh (help stabilization).
- The hidden state activation function is the identity.

sLSTM Forward Pass and Stabilization II

Exponential activation functions can lead to large values that cause overflow.

Stabilization equations:

$$
\begin{aligned}
\boxed{m_t} &= \max\left(\tilde{f}_t + \boxed{m_{t-1}}, \tilde{i}_t\right) && \text{stabilizer state} \\
i'_t &= \exp\left(\tilde{i}_t - \boxed{m_t}\right) && \text{stabilized input gate} \\
f'_t &= \exp\left(\tilde{f}_t + \boxed{m_{t-1}} - \boxed{m_t}\right) && \text{stabilized forget gate}
\end{aligned}
$$

Note: The stabilizer state m_t does not change network output nor gradients.

Proof of Equivalence for sLSTM Stabilized Version

$$c_t = c_t^{(s)} \exp(m_t)$$
$$n_t = n_t^{(s)} \exp(m_t)$$

Proof of Equivalence for sLSTM Stabilized Version

$$c_t = c_t^{(s)} \exp(m_t)$$
$$n_t = n_t^{(s)} \exp(m_t)$$

$$\begin{aligned}
\tilde{h}_t^{(s)} &= \frac{c_t^{(s)}}{n_t^{(s)}} \\
&= \frac{\exp(\log(f_t) + m_{t-1} - m_t) c_{t-1}^{(s)} + \exp(\log(i_t) - m_t) z_t}{\exp(\log(f_t) + m_{t-1} - m_t) n_{t-1}^{(s)} + \exp(\log(i_t) - m_t)} \\
&= \frac{\exp(\log(f_t) + m_{t-1}) c_{t-1}^{(s)} + \exp(\log(i_t)) z_t}{\exp(\log(f_t) + m_{t-1}) n_{t-1}^{(s)} + \exp(\log(i_t))} \\
&= \frac{\exp(\log(f_t)) c_{t-1} + \exp(\log(i_t)) z_t}{\exp(\log(f_t)) n_{t-1} + \exp(\log(i_t))} \\
&= \frac{f_t c_{t-1} + i_t z_t}{f_t n_{t-1} + i_t} = \frac{c_t}{n_t} = \tilde{h}_t
\end{aligned}$$

Memory Mixing in sLSTM vs. LSTM I

- **Standard LSTM**:
  - Recurrent weight matrices $(\mathbf{R}_z, \mathbf{R}_i, \mathbf{R}_f, \mathbf{R}_o)$ are full matrices.
  - Allows memory mixing across all memory cells.
  - **Example of a Full Recurrent Matrix** $(\mathbf{R} \in \mathbb{R}^{d \times d})$:

$$\mathbf{R} = \begin{pmatrix} r_{11} & r_{12} & \cdots & r_{1d} \\ r_{21} & r_{22} & \cdots & r_{2d} \\ \vdots & \vdots & \ddots & \vdots \\ r_{d1} & r_{d2} & \cdots & r_{dd} \end{pmatrix}$$

- **sLSTM with Multiple Heads**:
  - Recurrent weight matrices $(\mathbf{R}_z, \mathbf{R}_i, \mathbf{R}_f, \mathbf{R}_o)$ are **block-diagonal**.
  - Enables memory mixing within each head but not across heads.
  - **Number of heads**: N_h.
  - **Head size**: $d_h = \frac{d}{N_h}$.

Memory Mixing in sLSTM vs. LSTM II

- **Example of a Block-Diagonal Recurrent Matrix** ($\mathbf{R} \in \mathbb{R}^{d \times d}$):

$$\mathbf{R} = \begin{pmatrix} \mathbf{R}^{(1)} & \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{0} & \mathbf{R}^{(2)} & \dots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \dots & \mathbf{R}^{(N_h)} \end{pmatrix}$$

  where each $\mathbf{R}^{(k)} \in \mathbb{R}^{d_h \times d_h}$ corresponds to head k.

- **Parameter Reduction**:
  - **Standard LSTM**:

$$\text{Parameters} = d \times d = d^2$$

  - **sLSTM**:

$$\text{Parameters} = N_h \times d_h^2 = N_h \times \left(\frac{d}{N_h}\right)^2 = \frac{d^2}{N_h}$$

mLSTM: Enhanced Storage in LSTMs

- LSTMs use a scalar cell state $c_t \in \mathbb{R}$.
- **Goal**:
  - Increase storage capacity by extending the cell state to a matrix $\mathbf{C}_t \in \mathbb{R}^{d \times d}$.
  - Allow accumulation (storage) of information over time steps.
  - Enable retrieval of stored information **without** access to previous time steps.
- **Storing Key-Value Pairs and Retrieval**:
  - At each time step t, store:
    - **Key vector** $\mathbf{k}_t \in \mathbb{R}^d$.
    - **Value vector** $\mathbf{v}_t \in \mathbb{R}^d$.
  - Later retrieve $\mathbf{v}_t$ using a **query vector** $\mathbf{q}_{t+\tau}$.

Comparison with Attention Mechanisms

- At each time step, compute:
  - **Query vector $\mathbf{q}_t$.**
  - **Key vector $\mathbf{k}_t$.**
  - **Value vector $\mathbf{v}_t$.**
- **Attention Scores**:

$$\alpha_{t,t'} = \frac{\exp\left(\frac{\mathbf{q}_t^\top \mathbf{k}_{t'}}{\sqrt{d}}\right)}{\sum_{t''} \exp\left(\frac{\mathbf{q}_t^\top \mathbf{k}_{t''}}{\sqrt{d}}\right)}$$

- **Retrieval (Context Vector)**:

$$\mathbf{h}_t = \sum_{t'} \alpha_{t,t'} \mathbf{v}_{t'}$$

- The model attends to relevant parts of the input sequence by computing similarities.
- Allows parallel computation but requires access to **all** previous steps storage.

Outer Products and Bidirectional Associative Memories I

- **Outer Product and Rank-1 Matrices**:
  - The **outer product** of two vectors $\mathbf{u} \in \mathbb{R}^m$ and $\mathbf{v} \in \mathbb{R}^n$ is:

$$\mathbf{A} = \mathbf{u}\mathbf{v}^\top \in \mathbb{R}^{m \times n}$$

  - $\mathbf{A}$ is a **rank-1 matrix**:
    - All columns are scalar multiples of $\mathbf{u}$.
    - All rows are scalar multiples of $\mathbf{v}^\top$.
  - Represents all pairwise combinations between elements of $\mathbf{u}$ and $\mathbf{v}$.
- **Outer Product in Singular Value Decomposition (SVD)**:

$$\mathbf{A} = \sum_{i=1}^{r} \sigma_i \mathbf{u}_i \mathbf{v}_i^\top$$

  where:
    - r is the rank of $\mathbf{A}$.
    - σ_i are singular values ($\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r > 0$).
    - $\mathbf{u}_i \in \mathbb{R}^m$ and $\mathbf{v}_i \in \mathbb{R}^n$ are left and right singular vectors.

Outer Products and Bidirectional Associative Memories II

- **Outer Product in Associative Memories (BAM)**:
  - **BAM** stores associations between pairs $\{(\mathbf{x}_p, \mathbf{y}_p)\}$

$$\mathbf{W} = \sum_{p=1}^{P} \mathbf{y}_p \mathbf{x}_p^\top$$

  - Each outer product $\mathbf{y}_p \mathbf{x}_p^\top$ is a rank-1 matrix capturing the association between $\mathbf{x}_p$ and $\mathbf{y}_p$.
  - **Retrieval Mechanism**:

$$\mathbf{y}_{\text{retrieved}} = \mathbf{W}\mathbf{x}_q = \sum_{p=1}^{P} \mathbf{y}_p (\mathbf{x}_p^\top \mathbf{x}_q)$$

  - Inner product $\mathbf{x}_p^\top \mathbf{x}_q$ measures similarity. Retrieval amplifies matching patterns due to higher similarity.
  - Capacity depends on orthogonality of stored patterns.
  - Highly correlated vectors may cause interference and retrieval errors.

Outer Products and Bidirectional Associative Memories III

- **Example**:
  - **Stored Patterns**:

$$\mathbf{x}_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad \mathbf{y}_1 = \begin{bmatrix} a \\ b \end{bmatrix}$$

$$\mathbf{x}_2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \quad \mathbf{y}_2 = \begin{bmatrix} c \\ d \end{bmatrix}$$

  - **Weight Matrix**:

$$\mathbf{W} = \mathbf{y}_1 \mathbf{x}_1^\top + \mathbf{y}_2 \mathbf{x}_2^\top = \begin{bmatrix} a & c \\ b & d \end{bmatrix}$$

  - **Retrieval**:

$$\mathbf{y}_{\text{retrieved}} = \mathbf{W}\mathbf{x}_1 = \begin{bmatrix} a \\ b \end{bmatrix} = \mathbf{y}_1$$

$$\mathbf{y}_{\text{retrieved}} = \mathbf{W}\mathbf{x}_2 = \begin{bmatrix} c \\ d \end{bmatrix} = \mathbf{y}_2$$

Optimality of the Covariance Update Rule

- **Covariance Update Rule**:

$$\mathbf{C}_t = \mathbf{C}_{t-1} + (\mathbf{v}_t - \bar{\mathbf{v}})(\mathbf{k}_t - \bar{\mathbf{k}})^\top$$

- **Separability**:
  - This rule is optimal for maximal separability of retrieved binary vectors.
  - Higher separability is achievable when limiting retrieval to pairwise interactions.
  - Requires quadratic complexity, as in attention mechanisms.
- **Relation to Fast Weight Programmers**:
  - Covariance update rule is equivalent to Fast Weight Programmers (Schmidhuber, 1992; Schlag et al., 2021).
  - Incorporates dynamic weight updates for fast memory access.

Matrix LSTM (mLSTM) Equations

$$
\begin{array}{rcllr}
\boldsymbol{C}_t & = & \mathrm{f}_t \, \boldsymbol{C}_{t-1} + \mathrm{i}_t \, \boldsymbol{v}_t \, \boldsymbol{k}_t^\top & & \text{cell state} \\
\boldsymbol{n}_t & = & \mathrm{f}_t \, \boldsymbol{n}_{t-1} + \mathrm{i}_t \, \boldsymbol{k}_t & & \text{normalizer state} \\
\boldsymbol{h}_t & = & \mathbf{o}_t \odot \tilde{\boldsymbol{h}}_t \,, & \tilde{\boldsymbol{h}}_t = \boldsymbol{C}_t \, \boldsymbol{q}_t \, / \, \max\left\{ \left| \boldsymbol{n}_t^\top \, \boldsymbol{q}_t \right|, 1 \right\} & \text{hidden state} \\
\boldsymbol{q}_t & = & \boldsymbol{W}_q \, \boldsymbol{x}_t + \boldsymbol{b}_q & & \text{query input} \\
\boldsymbol{k}_t & = & \frac{1}{\sqrt{d}} \boldsymbol{W}_k \, \boldsymbol{x}_t + \boldsymbol{b}_k & & \text{key input} \\
\boldsymbol{v}_t & = & \boldsymbol{W}_v \, \boldsymbol{x}_t + \boldsymbol{b}_v & & \text{value input} \\
\mathrm{i}_t & = & \exp\left(\tilde{\mathrm{i}}_t\right) \,, & \tilde{\mathrm{i}}_t = \boldsymbol{w}_{\mathrm{i}}^\top \, \boldsymbol{x}_t + b_{\mathrm{i}} & \text{input gate} \\
\mathrm{f}_t & = & \sigma\left(\tilde{\mathrm{f}}_t\right) \text{ OR } \exp\left(\tilde{\mathrm{f}}_t\right) \,, & \tilde{\mathrm{f}}_t = \boldsymbol{w}_{\mathrm{f}}^\top \, \boldsymbol{x}_t + b_{\mathrm{f}} & \text{forget gate} \\
\mathbf{o}_t & = & \sigma\left(\tilde{\mathbf{o}}_t\right) \,, & \tilde{\mathbf{o}}_t = \boldsymbol{W}_{\mathbf{o}} \, \boldsymbol{x}_t + \boldsymbol{b}_{\mathbf{o}} & \text{output gate}
\end{array}
$$

$$
\text{LayerNorm}(\mathbf{x}) = \frac{\mathbf{x} - \mu}{\sigma} \odot \gamma + \beta
$$

Applied before projecting input to key and value (covariance updates)

Normalizer State and Numerical Stability

- **Normalizer State**:

$$\mathbf{n}_t = f_t \cdot \mathbf{n}_{t-1} + i_t \cdot \mathbf{k}_t$$

- **Purpose**:
  - Keeps a weighted sum of key vectors.
  - Each key vector is weighted by the input gate and all future forget gates.
  - Records the strength of the gates over time.
- **Numerical Stability**:
  - The dot product $\mathbf{n}_t^\top \mathbf{q}_t$ can be close to zero.
  - To prevent division by small numbers, use:

$$\tilde{h}_t = \frac{\mathbf{C}_t \mathbf{q}_t}{\max\left(\left|\mathbf{n}_t^\top \mathbf{q}_t\right|, 1\right)}$$

Multiple Memory Cells and Stabilization

- **Multiple Memory Cells**:
  - mLSTM can have multiple memory cells like the original LSTM.
  - For mLSTM, multiple heads and multiple cells are equivalent due to lack of memory mixing.
- **Parallelization** **Appendix A.3**:
  - Since mLSTM has no memory mixing, the recurrence can be reformulated in a parallel version.
  - Improves computational efficiency on GPUs.

mLSTM Computations: Iterative and Matrix Forms I

- **At each time step** t:

$$
\begin{aligned}
\text{Forget Gate:} \quad & f_t = \sigma(\tilde{f}_t) \\
\text{Input Gate:} \quad & i_t = \sigma(\tilde{i}_t) \\
\text{Output Gate:} \quad & o_t = \sigma(\tilde{o}_t) \\
\text{Cell State Update:} \quad & \mathbf{C}_t = f_t \mathbf{C}_{t-1} + i_t(\mathbf{v}_t \mathbf{k}_t^\top) \\
\text{Hidden State:} \quad & \mathbf{h}_t = o_t \widetilde{\mathbf{h}}_t
\end{aligned}
$$

- **Unrolling the Cell State**:

$$
\begin{aligned}
\mathbf{C}_t &= f_t f_{t-1} \mathbf{C}_{t-2} + f_t i_{t-1}(\mathbf{v}_{t-1} \mathbf{k}_{t-1}^\top) + i_t(\mathbf{v}_t \mathbf{k}_t^\top) \\
&= \left(\prod_{k=1}^{t} f_k \right) \mathbf{C}_0 + \sum_{j=1}^{t} \left(\left(\prod_{k=j+1}^{t} f_k \right) i_j(\mathbf{v}_j \mathbf{k}_j^\top) \right)
\end{aligned}
$$

- **Assuming** $\mathbf{C}_0 = \mathbf{0}$ for simplicity.

mLSTM Computations: Iterative and Matrix Forms II

- **Constructing the Forget Gate Activation Matrix F**:

$$\tilde{\mathbf{f}} = [\tilde{f}_1, \tilde{f}_2, \ldots, \tilde{f}_T]^\top \in \mathbb{R}^T$$

$$\mathbf{F}_{ij} = \begin{cases} 0 & \text{for } j > i \\ 1 & \text{for } j = i \\ \prod_{k=j+1}^{i} \sigma(\tilde{f}_k) & \text{for } j < i \end{cases}$$

- **Visualization of F**:

$$\mathbf{F} = \begin{bmatrix} 1 & 0 & \ldots & 0 \\ \sigma(\tilde{f}_2) & 1 & \ldots & 0 \\ \sigma(\tilde{f}_2)\sigma(\tilde{f}_3) & \sigma(\tilde{f}_3) & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ \prod_{k=2}^{T} \sigma(\tilde{f}_k) & \prod_{k=3}^{T} \sigma(\tilde{f}_k) & \ldots & 1 \end{bmatrix}$$

mLSTM Computations: Iterative and Matrix Forms III

- **Constructing the Input Gate Pre-Activation Matrix $\tilde{\mathbf{I}}$:**

$$\tilde{\mathbf{i}} = [\tilde{i}_1, \tilde{i}_2, \ldots, \tilde{i}_T]^\top \in \mathbb{R}^T$$

$$\tilde{\mathbf{I}}_{ij} = \begin{cases} 0 & \text{for } j > i \\ \tilde{i}_j & \text{for } i \geq j \end{cases}$$

- **Computing the Unstabilized Gate Activation Matrix $\mathbf{D}$:**

$$\mathbf{D} = \mathbf{F} \odot \exp\left(\tilde{\mathbf{I}}\right)$$

- **Computing Hidden Pre-Activation States $\widetilde{\mathbf{H}}$:**

$$\widetilde{\mathbf{C}} = \frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d}} \odot \mathbf{D}$$

$$\mathbf{C}_i = \frac{\widetilde{\mathbf{C}}_i}{\max\left(\left|\sum_{j=1}^{T} \widetilde{\mathbf{C}}_{ij}\right|, 1\right)}$$

$$\widetilde{\mathbf{H}} = \mathbf{C}\mathbf{V}$$

mLSTM Computations: Iterative and Matrix Forms IV

- **Computing the Final Hidden States $\mathbf{H}$**:

$$\tilde{\mathbf{O}} \in \mathbb{R}^{T \times d}, \quad \mathbf{O} = \sigma(\tilde{\mathbf{O}})$$

$$\mathbf{H} = \mathbf{O} \odot \widetilde{\mathbf{H}}$$

- **Equivalence Between Iterative and Matrix Forms**:
  - **Iterative Computation**:

$$\mathbf{C}_t = f_t \mathbf{C}_{t-1} + i_t (\mathbf{v}_t \mathbf{k}_t^\top)$$

  - **Matrix Formulation**:

$$\mathbf{C} = \mathbf{F} \odot \exp\left(\tilde{\mathbf{I}}\right) \odot (\text{Outer Products of } \mathbf{v}_t \mathbf{k}_t^\top)$$

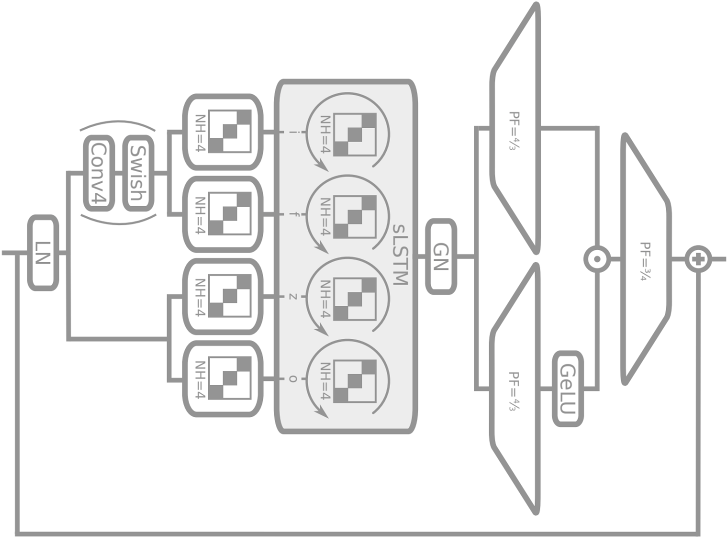
  - **Explanation**:
    - Matrix operations aggregate iterative updates.
    - Enables parallel computation over the entire sequence.

xLSTM Architecture

- xLSTM Blocks: Designed to non-linearly summarize the past in a high-dimensional space, enhancing the separation of different histories or contexts (Cover's Theorem (1965)).
- Residual Block Architectures:
  - Post Up-Projection (like Transformers):
    - Input is fed into an sLSTM, optionally followed by a convolution.
    - A gated MLP follows the sLSTM block.
  - Pre Up-Projection (like State Space Models):
    - Input is mapped into a high-dimensional space and linearly maps back after non-linear summarization.
    - mLSTM is wrapped inside two MLPs, with a convolution, skip connection, and an output gate.
- Construction: Residual stacking with pre-LayerNorm, as used in large language models.

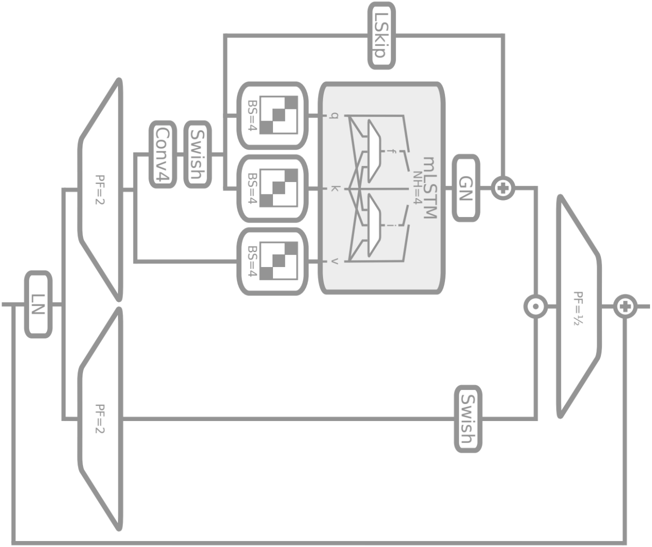
Post Up-projection Residual Block

Mainly for sLSTM



Pre Up-projection Residual Block

Mainly for xLSTM



Memory and Speed Considerations

- Linear Computation and Constant Memory Complexity:
  - xLSTM offers linear computational complexity and constant memory complexity with respect to sequence length.
  - This makes xLSTM suitable for industrial applications and on-edge implementations.
- mLSTM Memory and Computational Trade-Off:
  - Memory is a $d \times d$ matrix, which is parameter-free but computationally intensive.
  - Trade-off between memory capacity and computational complexity, manageable with parallel GPU computations.
- Parallelization:
  - mLSTM is parallelizable (similar to FlashAttention).
  - sLSTM is not parallelizable due to memory mixing but is optimized with fast CUDA implementation.

Test of xLSTM's Exponential Gating with Memory Mixing

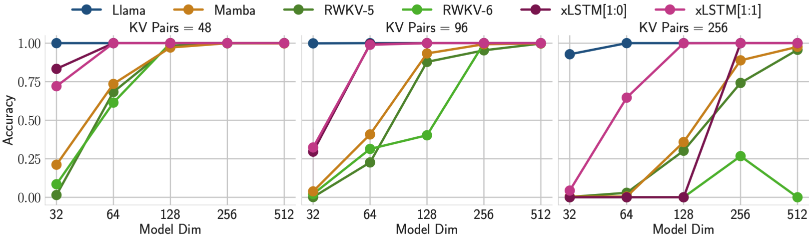
```
Bucket Sort
  Sequence: 1 4 8 6 1 1 1 4 6 8
Cycle Nav
  Sequence: STAY +1 -1 +1 STAY +1 +1 +1 -1 P3
Even Pairs
  Sequence: a b b a a b a b a a
Majority
  Sequence: 1 7 6 4 3 8 1 7 2 1
Majority Count
  Sequence: 1 7 6 4 4 8 1 7 2 2
Missing Duplicate
  Sequence: 4 8 6 2 5 4 8 6 2 [MIS] 5
Mod Arithmetic (w/o Braces)
  Sequence: 0 - 4 + 0 - 2 = 4 [PAD]
Mod Arithmetic (w Braces)
  Sequence: ( ( ( 2 ) * - 2 ) - ( - 4 - 2 ) ) = 2
Odds First
  Sequence: 2 7 3 2 6 9 [ACT] 2 3 6 7 2 9
Parity:
  Sequence: a b b a a b a b
Repetition
  Sequence: 2 4 8 6 2 [ACT] 2 4 8 6 2
Reverse String
  Sequence: 2 4 8 6 2 [ACT] 2 6 8 4 2
Stack Manipulation
  Sequence: ST1 ST1 ST3 POP POP PS3 PS3 [ACT] ST1 ST3 ST3
Set
  Sequence: 8 6 6 3 5 4 5 3 [ACT] 8 6 3 5 4
Solve Equation:
  Sequence: ( ( ( 2 + 0 ) + - x ) - ( 1 ) ) = 2 [ACT] 2
```

Test of xLSTM's Exponential Gating with Memory Mixing

	Context Sentsitive		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 ± 0.01	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 ± 0.21	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

Test of xLSTM's Memory Capacities on Associative Recall Tasks

Test memory capacity on the Multi-Query Associative Recall task: memorizing randomly chosen key-value pairs for later retrieval, 256 pairs, context length is 2048.



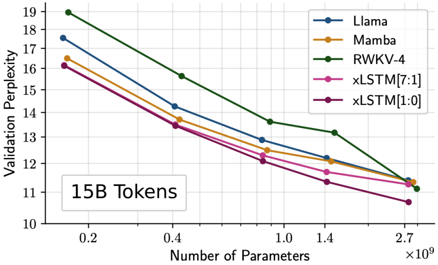
Test of xLSTM's Long Context Capabilities on Long Range Arena

	Retrieval acc ↑	ListOps acc ↑	Pathfinder acc ↑	G-Image acc ↑	RGB-Image acc ↑	Ranking acc ↑
Random Baseline	0.500	0.100	0.500	0.100	0.100	
Llama	0.845	0.379	0.887	0.541	0.629	5.2
Mamba	<u>0.902</u>	0.325	**0.992**	0.689	**0.765**	2.2
RWKV-4	0.898	0.389	0.914	<u>0.691</u>	0.757	3.0
LSTM	X	0.275	X	0.675	0.718	5.4
LSTM (Block)	0.880	**0.495**	X	0.690	0.756	3.4
xLSTM	**0.906**	<u>0.411</u>	<u>0.919</u>	**0.695**	<u>0.761</u>	1.6

Perplexity

Train an auto-regressive language model on 15B tokens from SlimPajama

Model	#Params M	SlimPajama (15B) ppl ↓
GPT-3	356	14.26
Llama	407	14.25
H3	420	18.23
Mamba	423	13.70
Hyena	435	17.59
RWKV-4	430	15.62
RWKV-5	456	16.53
RWKV-6	442	17.40
RetNet	431	16.23
HGRN	411	21.83
GLA	412	19.56
HGRN2	411	16.77
xLSTM[1:0]	409	**13.43**
xLSTM[7:1]	408	13.48



Ablation

Ablation studies on the new xLSTM components.

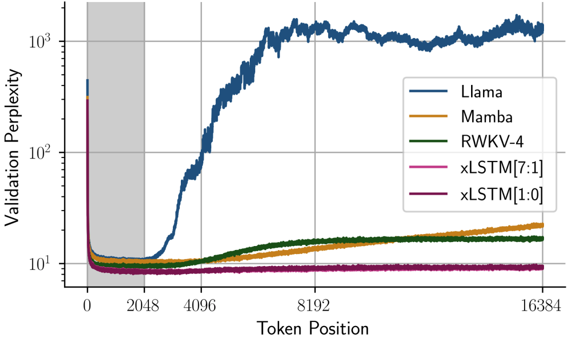
Model	Modification	Exponential Gating	Matrix Memory	#Params M	SlimPajama (15B) ppl ↓
LSTM	Vanilla Multi-Layer LSTM	✗	✗	607.8	2417.86
	Adding Resnet Backbone	✗	✗	506.1	35.46
	Adding Up-Projection Backbone	✗	✗	505.9	26.01
xLSTM[0:1]	Adding Exponential Gating	✓	✗	427.3	17.70
xLSTM[7:1]	Adding Matrix Memory	✓	✓	408.4	**13.48**

Ablation studies on different gating techniques.

Learnable Gates	Forget Gate			Input Gate			SlimPajama (15B) ppl ↓
	Input Dependent	Learnable Bias	Bias Init	Input Dependent	Learnable Bias	Bias Init	
No Gates	✗	✗	$+\infty$	✗	✗	0	NaN
No Gates	✗	✗	$[3,6]$	✗	✗	0	13.95
Forget Gate	✓	✓	$[3,6]$	✗	✗	0	13.58
Input Gate	✗	✗	$[3,6]$	✓	✓	$\mathcal{N}(0,0.1)$	13.69
Forget Gate Bias	✗	✓	$[3,6]$	✗	✗	0	13.76
Forget + Input Gate Bias	✗	✓	$[3,6]$	✗	✓	$\mathcal{N}(0,0.1)$	13.73
Forget Gate + Input Gate Bias	✓	✓	$[3,6]$	✗	✓	$\mathcal{N}(0,0.1)$	13.55
Forget Gate + Input Gate	✓	✓	$[3,6]$	✓	✓	$\mathcal{N}(0,0.1)$	**13.43**

Sequence Length Extrapolation

Training on 300B tokens, model size 1.3B



Model	SlimPajama (300B) ppl ↓ at 16k
Llama	337.83
Mamba	14.00
RWKV-4	13.75
xLSTM[7:1]	**8.92**
xLSTM[1:0]	9.01

Validation Perplexity and Downstream Tasks.

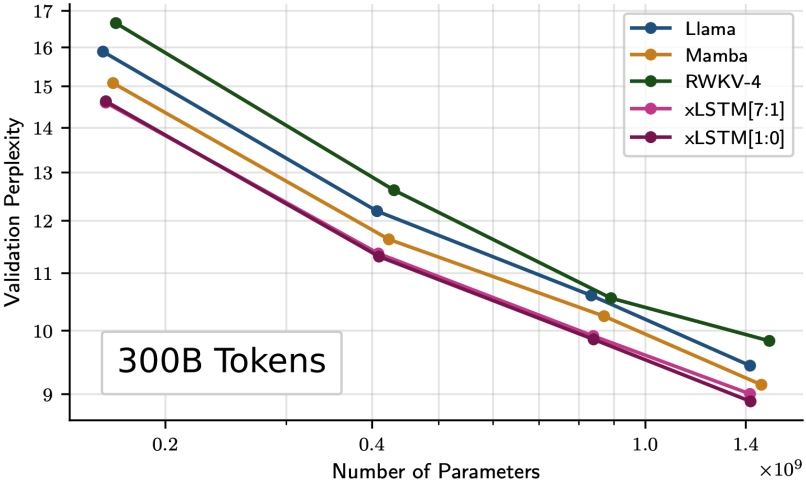
Training on 300B tokens, model sizes (125M, 350M, 760M, 1.3B)

	Model	#Params M	SlimPajama (300B) ppl ↓	LAMBADA ppl ↓	LAMBADA acc ↑	HellaSwag acc ↑	PIQA acc ↑	ARC-E acc ↑	ARC-C acc ↑	WinoGrande acc ↑	Average acc ↑
125M	RWKV-4	169.4	16.66	54.72	23.77	34.03	66.00	47.94	24.06	50.91	41.12
	Llama	162.2	15.89	39.21	31.54	34.09	65.45	45.33	23.63	50.67	41.78
	Mamba	167.8	15.08	27.76	34.14	36.47	66.76	**48.86**	24.40	51.14	43.63
	xLSTM[1:0]	163.8	14.63	**25.98**	**36.52**	36.74	65.61	47.81	24.83	**51.85**	43.89
	xLSTM[7:1]	163.7	**14.60**	26.59	36.08	**36.75**	**66.87**	48.32	**25.26**	51.70	**44.16**
350M	RWKV-4	430.5	12.62	21.57	36.62	42.47	69.42	54.46	25.43	51.22	46.60
	Llama	406.6	12.19	15.73	44.19	44.45	69.15	52.23	26.28	53.59	48.32
	Mamba	423.1	11.64	12.83	46.24	47.55	69.70	55.47	27.56	54.30	50.14
	xLSTM[1:0]	409.3	**11.31**	**11.49**	**49.33**	**48.06**	69.59	55.72	26.62	**54.38**	50.62
	xLSTM[7:1]	408.4	11.37	12.11	47.74	47.89	**71.16**	**56.61**	27.82	53.28	**50.75**
760M	RWKV-4	891.0	10.55	10.98	47.43	52.29	72.69	58.84	28.84	55.41	52.58
	Llama	834.1	10.60	9.90	51.41	52.16	70.95	56.48	28.75	56.67	52.74
	Mamba	870.5	10.24	9.24	50.84	53.97	71.16	60.44	29.78	56.99	53.86
	xLSTM[1:0]	840.4	**9.86**	8.09	54.78	55.72	72.69	**62.75**	**32.59**	**58.17**	**56.12**
	xLSTM[7:1]	839.7	9.91	**8.07**	**55.27**	**56.12**	**72.74**	61.36	29.61	56.43	55.26
1.3B	RWKV-4	1515.2	9.83	9.84	49.78	56.20	74.70	61.83	30.63	55.56	54.78
	Llama	1420.4	9.44	7.23	57.44	57.81	73.12	62.79	31.74	59.04	56.99
	Mamba	1475.3	9.14	7.41	55.64	60.45	74.43	**66.12**	**33.70**	60.14	58.41
	xLSTM[1:0]	1422.6	**8.89**	**6.86**	**57.83**	**60.91**	74.59	64.31	32.59	**60.62**	**58.48**
	xLSTM[7:1]	1420.1	9.00	7.04	56.69	60.26	**74.92**	65.11	32.34	59.27	58.10

Performance on PALOMA Language Tasks

	Model	#Params M	C4	MC4 EN	Wikitext 103	Penn Treebank	Red Pajama	Refined Web	Dolma	M2D2 S2ORC	M2D2 Wikipedia	C4 Domains	Dolma Subreddits	Dolma Coding	Average
125M	RWKV-4	169.4	26.25	22.33	29.18	38.45	8.99	32.47	17.04	23.86	21.42	22.68	37.08	5.12	23.74
	Llama	162.2	24.64	17.23	23.16	31.56	8.26	29.15	15.10	19.71	20.41	21.45	36.73	3.61	20.92
	Mamba	167.8	23.12	17.04	22.49	30.63	7.96	27.73	14.60	19.38	19.36	20.14	34.32	3.77	20.05
	xLSTM[1:0]	163.8	22.54	16.32	21.98	30.47	7.80	27.21	14.35	19.02	19.04	19.65	34.15	3.64	19.68
	xLSTM[7:1]	163.7	**22.39**	**16.13**	**21.47**	**30.01**	**7.75**	**26.91**	**14.13**	**18.6**	**18.84**	**19.52**	**33.9**	**3.59**	**19.44**
350M	RWKV-4	430.5	19.55	15.82	19.64	27.58	6.97	24.28	12.94	17.59	15.96	16.98	29.40	3.90	17.55
	Llama	406.6	18.38	13.28	16.41	**21.82**	6.56	22.09	11.76	15.05	15.25	15.99	28.30	3.12	15.67
	Mamba	423.1	17.33	13.05	16.11	22.24	6.34	21.04	11.42	14.83	14.53	15.16	27.02	3.20	15.19
	xLSTM[1:0]	409.3	17.01	**12.55**	**15.17**	22.51	**6.20**	**20.66**	**11.16**	**14.44**	**14.27**	**14.85**	26.70	**3.08**	**14.88**
	xLSTM[7:1]	408.4	**16.98**	12.68	15.43	21.86	6.23	20.70	11.22	14.62	14.30	**14.85**	**26.61**	3.11	**14.88**
760M	RWKV-4	891.0	15.51	12.76	14.84	21.39	5.91	19.28	10.70	14.27	13.04	13.68	24.22	3.32	14.08
	Llama	834.1	15.75	11.59	13.47	18.33	5.82	19.04	10.33	13.00	13.05	13.76	24.80	2.90	13.49
	Mamba	870.5	15.08	11.54	13.47	19.34	5.69	18.43	10.15	13.05	12.62	13.25	23.94	2.99	13.30
	xLSTM[1:0]	840.4	**14.60**	**11.03**	**12.61**	17.74	**5.52**	**17.87**	**9.85**	**12.50**	**12.20**	**12.81**	23.46	**2.87**	**12.76**
	xLSTM[7:1]	839.7	14.72	11.11	12.68	**17.61**	5.55	18.01	9.87	12.59	12.25	12.89	**23.43**	2.88	12.80
1.3B	RWKV-4	1515.2	14.51	12.04	13.73	19.37	5.62	18.25	10.11	13.46	12.10	12.87	22.85	3.25	13.18
	Llama	1420.4	13.93	10.44	11.74	**15.92**	5.29	17.03	9.35	11.61	11.53	12.24	22.63	2.74	12.04
	Mamba	1475.3	13.35	10.40	11.76	16.65	5.21	16.50	9.17	11.73	11.18	11.83	21.43	2.83	11.84
	xLSTM[1:0]	1422.6	**13.13**	**10.09**	11.41	**15.92**	**5.10**	**16.25**	**9.01**	**11.43**	**10.95**	**11.60**	**21.29**	**2.73**	**11.58**
	xLSTM[7:1]	1420.1	13.31	10.21	**11.32**	16.00	5.16	16.48	9.11	11.61	11.10	11.76	21.50	2.75	11.69

Scaling Laws



Thanks!