Assignment: LSTM and GRU vs. Multiplicative Variants on the Copy Task

Objective.
In this assignment, you will modify RNN cell shared in the class (Please refer to the jupyter notebook on the GitHub page). You have flexibility to choose from TF, Jax or Pytorch. But you will only modify the given notebook. :

1. Implement four recurrent neural network architectures:
   - Standard LSTM
   - Multiplicative LSTM
   - Standard GRU
   - Multiplicative GRU
2. Evaluate their ability to perform the **copy task** with increasing sequence lengths $\{100, 200, 500, 1000\}$. Remember to construct your train, test and validation splits.
3. Compare each model's *long-term memory capability*, convergence speed, and final test accuracy.

1. The Copy Task
The copy task checks if a model can *remember and reproduce* a sequence after a certain delay:

- Generate a sequence of length T of random symbols (e.g., from a small vocabulary of 8–10 tokens).
- Present the sequence to the model, then include a delimiter (or blank token) for several steps.
- After the delimiter, the model must *output* (copy) the original sequence of length T.

You will test on lengths $T \in \{100, 200, 500, 1000\}$. As T grows, the task requires the model to hold information in memory for a correspondingly longer time, highlighting differences in gating mechanisms.

2. Model Equations

Below are the equations for each architecture. Implement them exactly as described, or use built-in framework modules when possible (e.g., PyTorch's LSTM/GRU) but modify for multiplicative gating.

2.1 Standard LSTM

$$
\begin{aligned}
\mathbf{i}_t &= \sigma\big(\mathbf{W}^{(i)}\mathbf{x}_t + \mathbf{U}^{(i)}\mathbf{h}_{t-1} + \mathbf{b}^{(i)}\big),\\
\mathbf{f}_t &= \sigma\big(\mathbf{W}^{(f)}\mathbf{x}_t + \mathbf{U}^{(f)}\mathbf{h}_{t-1} + \mathbf{b}^{(f)}\big),\\
\mathbf{o}_t &= \sigma\big(\mathbf{W}^{(o)}\mathbf{x}_t + \mathbf{U}^{(o)}\mathbf{h}_{t-1} + \mathbf{b}^{(o)}\big),\\
\tilde{\mathbf{c}}_t &= \tanh\big(\mathbf{W}^{(c)}\mathbf{x}_t + \mathbf{U}^{(c)}\mathbf{h}_{t-1} + \mathbf{b}^{(c)}\big),\\
\mathbf{c}_t &= \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tilde{\mathbf{c}}_t,\\
\mathbf{h}_t &= \mathbf{o}_t \odot \tanh(\mathbf{c}_t).
\end{aligned}
$$

2.2 Multiplicative LSTM

Introduce a learnable *memory matrix* $\mathbf{M}_t$ or vector $\mathbf{m}_t$:

$$
\mathbf{m}_t = \mathbf{W}_m\,\mathbf{x}_t + \mathbf{U}_m\,\mathbf{h}_{t-1} + \mathbf{b}_m, \quad \mathbf{M}_t = \operatorname{diag}(\mathbf{m}_t),
$$

and define

$$
\tilde{\mathbf{x}}_t = \mathbf{M}_t\,\mathbf{x}_t \;\text{ or }\; \tilde{\mathbf{x}}_t = \mathbf{m}_t \odot \mathbf{x}_t.
$$

Then substitute $\tilde{\mathbf{x}}_t$ in place of $\mathbf{x}_t$ in the LSTM equations:

$$
\begin{aligned}
\mathbf{i}_t &= \sigma\big(\mathbf{W}^{(i)}\tilde{\mathbf{x}}_t + \mathbf{U}^{(i)}\mathbf{h}_{t-1} + \mathbf{b}^{(i)}\big),\\
\mathbf{f}_t &= \sigma\big(\mathbf{W}^{(f)}\tilde{\mathbf{x}}_t + \mathbf{U}^{(f)}\mathbf{h}_{t-1} + \mathbf{b}^{(f)}\big),\\
\mathbf{o}_t &= \sigma\big(\mathbf{W}^{(o)}\tilde{\mathbf{x}}_t + \mathbf{U}^{(o)}\mathbf{h}_{t-1} + \mathbf{b}^{(o)}\big),\\
\tilde{\mathbf{c}}_t &= \tanh\big(\mathbf{W}^{(c)}\tilde{\mathbf{x}}_t + \mathbf{U}^{(c)}\mathbf{h}_{t-1} + \mathbf{b}^{(c)}\big),\\
\mathbf{c}_t &= \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tilde{\mathbf{c}}_t,\\
\mathbf{h}_t &= \mathbf{o}_t \odot \tanh(\mathbf{c}_t).
\end{aligned}
$$

2.3 Standard GRU

$$
\begin{aligned}
\mathbf{z}_t &= \sigma\big(\mathbf{W}^{(z)}\mathbf{x}_t + \mathbf{U}^{(z)}\mathbf{h}_{t-1} + \mathbf{b}^{(z)}\big),\\
\mathbf{r}_t &= \sigma\big(\mathbf{W}^{(r)}\mathbf{x}_t + \mathbf{U}^{(r)}\mathbf{h}_{t-1} + \mathbf{b}^{(r)}\big),\\
\tilde{\mathbf{h}}_t &= \tanh\big(\mathbf{W}^{(h)}\mathbf{x}_t + \mathbf{U}^{(h)}(\mathbf{r}_t \odot \mathbf{h}_{t-1}) + \mathbf{b}^{(h)}\big),\\
\mathbf{h}_t &= (1 - \mathbf{z}_t) \odot \mathbf{h}_{t-1} + \mathbf{z}_t \odot \tilde{\mathbf{h}}_t.
\end{aligned}
$$

2.4 Multiplicative GRU

Again, define

$$\mathbf{m}_t = \mathbf{W}_m \mathbf{x}_t + \mathbf{U}_m \mathbf{h}_{t-1} + \mathbf{b}_m, \quad \tilde{\mathbf{x}}_t = \mathbf{m}_t \odot \mathbf{x}_t \;\text{ or }\; \tilde{\mathbf{x}}_t = \mathbf{M}_t \, \mathbf{x}_t.$$

Then substitute $\tilde{\mathbf{x}}_t$:

$$\begin{aligned}
\mathbf{z}_t &= \sigma\Big(\mathbf{W}^{(z)} \, \tilde{\mathbf{x}}_t + \mathbf{U}^{(z)} \mathbf{h}_{t-1} + \mathbf{b}^{(z)}\Big), \\
\mathbf{r}_t &= \sigma\Big(\mathbf{W}^{(r)} \, \tilde{\mathbf{x}}_t + \mathbf{U}^{(r)} \mathbf{h}_{t-1} + \mathbf{b}^{(r)}\Big), \\
\tilde{\mathbf{h}}_t &= \tanh\Big(\mathbf{W}^{(h)} \, \tilde{\mathbf{x}}_t + \mathbf{U}^{(h)} (\mathbf{r}_t \odot \mathbf{h}_{t-1}) + \mathbf{b}^{(h)}\Big), \\
\mathbf{h}_t &= (1 - \mathbf{z}_t) \odot \mathbf{h}_{t-1} + \mathbf{z}_t \odot \tilde{\mathbf{h}}_t.
\end{aligned}$$

3. Experimental Setup for the Copy Task

1. **Sequence Lengths:** $\{100, 200, 500, 1000\}$.
2. **Data Generation:**
   - Create random sequences of length T from a small discrete vocabulary (e.g. 8–10 symbols).
   - Insert a delimiter for several steps, after which the model should reproduce the original sequence.
3. **Model Training:**
   - Use the same hyperparameters (hidden size, batch size, learning rate, etc.) for all four architectures to ensure a fair comparison.
   - Train each model to minimize cross-entropy loss on the *predicted symbols* vs. the *target* symbols.
4. **Evaluation:**
   - Evaluate final accuracy on held-out sequences at each T.
   - Record how well each model copies the sequence with increasing delay (T).
   - Report your mean performance and standard error over 3 trials

4. Deliverables

1. **Code Implementation:**

- Scripts/notebooks that define all four architectures and generate data for the copy task.
- Training loops with clear logging of loss and accuracy.

2. **Results and Analysis:**
   - Final test accuracy or per-timestep accuracy for $T \in \{100, 200, 500, 1000\}$.
   - Plots of training/validation curves (loss and accuracy vs. epochs).
   - Discussion of which model handles the longest sequence best, and speculation on why.
3. **Observations on Multiplicative Effects:**
   - Comment on whether you observe easier long-term gradient flow or faster convergence in the multiplicative versions.
   - Note any drawbacks (e.g. training instability, hyperparameter sensitivity).

5. Grading Criteria

- **Correctness (40%):** Properly implemented equations for each architecture (LSTM, GRU, and their multiplicative counterparts).
- **Experimental Rigor (30%):** Fair comparisons (identical hyperparams), thorough evaluation, and clear reporting of metrics.
- **Analysis and Discussion (20%):** Quality of insights regarding long-term memory, gating behavior, and model stability.
- **Code Quality (10%):** Readable, well-documented code and reproducible experiments (with `README` or instructions).

References:

1. S. Hochreiter & J. Schmidhuber, "Long Short-Term Memory," *Neural Computation*, 9(8):1735–1780, 1997.
2. K. Cho *et al.*, "Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation," in *EMNLP*, 2014.
3. V. Mnih *et al.*, "Multiplicative interactions and their application to RNNs," *arXiv preprint*, 2016.