Thanks to the reviewers for their valuable feedback. We are encouraged by their endorsements that our MomentumRNN framework: 1) makes a very novel and thought-provoking [R3] connection between RNN and optimization [R3,R4], 2) provides a thorough analysis integrating various forms of momentum into different RNN types [R2], and 3) leads to decent improvements over baselines [R1,R2,R3]. Below we address the concerns raised by the reviewers.

[R1,R4] Momentum cell is similar to gated update $\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \mathbf{h}_t$ of LSTMs and residual mappings. We respectfully disagree with this point. We believe there is a misunderstanding. The MomentumRNN cell updates as: $\mathbf{v}_t = \mu\mathbf{v}_{t-1} + s\mathbf{W}\mathbf{x}_t; \mathbf{h}_t = \sigma(\mathbf{U}\mathbf{h}_{t-1} + \mathbf{v}_t)$, which introduces an auxiliary state $\mathbf{v}_t$, inspired by Nesterov momentum. First, for the hidden state dynamics, the two-step update in momentum cell is equivalent to $\mathbf{h}_t = \sigma(\mathbf{U}(\mathbf{h}_{t-1} - \mu\mathbf{h}_{t-2}) + \mu\sigma^{-1}(\mathbf{h}_{t-1}) + s\mathbf{W}\mathbf{x}_t)$, i.e., $\mathbf{h}_t$ directly depends on $\mathbf{h}_{t-1}$ and $\mathbf{h}_{t-2}$; in LSTM, $\mathbf{h}_t$ only directly depends on $\mathbf{h}_{t-1}$. $\mu\sigma^{-1}(\mathbf{h}_{t-1})$ is the key term that helps alleviate the vanishing gradient. Second, $\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \mathbf{h}_t$ differs from $\mathbf{v}_t = \mu\mathbf{v}_{t-1} + s\mathbf{W}\mathbf{x}_t$, where $\mathbf{f}$ and $\mathbf{i}$ are tensors while μ and s are scalars; also, $\mathbf{h}_t$ is the hidden state while $\mathbf{x}_t$ is the input. Third, $\mathbf{v}_t = \mu\mathbf{v}_{t-1} + s\mathbf{W}\mathbf{x}_t$ differs from residual mapping ($\mathbf{v}_t = \mathbf{v}_{t-1} + \mathcal{F}(\mathbf{v}_{t-1})$). As pointed out by [R3], "*In LSTM, the cell state performs additive integration of the input so that the gradients do not vanish. However, LSTM's sigmoid and tanh gates make gradients vanish. MomentumRNN performs additive integration of the input without any additional gates and hence has a much better gradient flow.*" We hope reviewers can reevaluate this crucial point.

[R1] Analogy between MomentumRNN and optimization methods. We do not aim to equate RNNs with gradient descent. Rather, we aim to bring in new ideas from optimization to design better RNNs. The nonlinear activation can be considered as a projection, and, based on our experiments, the acceleration from momentum is preserved. Moreover, the resulting momentum cell derived from optimization algorithms with momentum can alleviate the vanishing gradient.

[R1] Analysis is w.r.t. the vanilla RNN. Also, it is not obvious that an appropriate μ exists. In Sec. 2.3, we proved that $\mu\sigma^{-1}(\mathbf{h}_{t-1})$ in the MomentumRNN cell $\mathbf{h}_t = \sigma(\mathbf{U}(\mathbf{h}_{t-1} - \mu\mathbf{h}_{t-2}) + \mu\sigma^{-1}(\mathbf{h}_{t-1}) + s\mathbf{W}\mathbf{x}_t)$ is key to alleviate the vanishing gradient in vanilla RNNs, which generalizes to the analysis of other RNNs. Our empirical study in Fig. 2 is not for a vanilla RNN, but for the SOTA DTRIV model (see Appendix A.4). Our experiments show that MomentumRNNs significantly outperform all studied baseline RNN models, which empirically validates that an appropriate μ exists.

[R1,R4] Toy datasets and lack of comparison to other SOTA RNN baselines. We used the (P)MNIST, TIMIT, and Penn TreeBank (PTB) benchmarks, which are not toy. Our momentum-based method can be applied to many other RNNs, including those that that address vanishing gradients, to improve their performance. In Sec. 3 and 4, we have shown this in the case of LSTM and expRNN (DTRIV), a SOTA RNN model. Our approach outperforms both.

[R1,R3] More related work on long-term dependencies and saturations of LSTM. We appreciate the suggestions and will add JANET, NRU, and more papers on alleviating long terms dependency issues in the revision.

[R2] Why the MomentumRNN provides improvements over RNN is not clear. MomentumRNN mitigates vanishing gradients as analyzed in Sec 2.3. The reason it can theoretically accelerate convergence, improve robustness, and lead to higher performance is under our study. In particular, we are studying the continuum limit of the MomentumRNN.

[R2] Compare the computation cost of MomentumRNN & RNN to reach similar acc. When training on the PMNIST task using 256 hidden units, we observe that *to reach 92.29% test acc. for LSTM (see Tab. 1), LSTM needs* **699min** *while MomentumLSTM & RMSPropLSTM (our best model for this task) only need* **416min** *and* **403min***, resp.*

[R3] Why should RMSPropLSTM work? RMSPropLSTM inherits its adaptive step size from RMSProp. In training, $\mathbf{W}\mathbf{x}_t$ is rescaled adaptively and can improve training and enhance performance for certain tasks.

[R3] Why using different optimizers for different tasks? Results on using Adam for all models. We used the default optimizers that achieve SOTA results for different tasks, e.g., we used *RMSProp for (P)MNIST, Adam for TIMIT, and SGD for PTB*. Using Adam to train LSTM, MomentumLSTM, AdamLSTM, RMSPropLSTM, and SRLSTM with 256 hidden units on PMNIST, we obtain test accuracy of $92.05 \pm 0.63\%$, $92.47 \pm 0.35\%$, $92.53 \pm 0.26\%$, $93.86 \pm 0.24\%$ and $92.17 \pm 1.37\%$, resp. Adam trainings for DTRIV and MomentumDTRIV with 512 hidden units on PMNIST yield the test accuracy of $95.78 \pm 0.21\%$ and $96.01 \pm 0.10\%$, resp. Adam training yields worse results than SGD on PTB.

[R3] 1) Why using only the MomentumRNN in language modelling task? 2) Fig. 6 shows that often momentum=0 is the best thing to do? 3) Why is forget gate initialized to -4? 1) We compare 3-layer MomentumLSTM with 3-layer LSTM in our paper (see Tab. 3). The test PPL for AdamLSTM, RMSPropLSTM, and SRLSTM are 61.11 ± 0.31, 64.53 ± 0.20, and 58.83 ± 0.62, resp. **2)** In Fig. 6, momentum=0 yields the best results only for language modeling task. We believe that if we do finer-scale search for momentum and step size, we can obtain a better result with a non-zero momentum. **3)** For the TIMIT task, initializing the forget gate bias to -4 is suggested in [arXiv:1707.09520].

[R2,R4] Additional hyper-parameters & their sensitivity. We search μ & s independently for a few options which is not very sensitive and does not increase the computational cost much. Also, trainable parameters is under our study.

[R4] Copy/adding tasks. In copying (adding) task for sequences of length 2K (750), our MomentumLSTM, AdamLSTM, RMSPropLSTM, and SRLSTM achieve the final training loss of 0.009 (0.162), **0.004** (0.006), 0.008 (**0.004**), and 0.01 (0.166), resp. while the training loss of the baseline LSTM is 0.01 (0.162). AdamLSTM and RMSPropLSTM converge faster than LSTM in both tasks. Moreover, AdamDTRIV and RMSPropDTRIV converge to the same final training loss remarkably faster than DTRIV in both tasks. Detailed results are included in the revision.