Understanding and Controlling Memory in Recurrent Neural Networks

Doron Haviv [1 2] **Alexnadre Rivkind** [2 3 4] **Omri Barak** [2 3]

Abstract

To be effective in sequential data processing, Recurrent Neural Networks (RNNs) are required to keep track of past events by creating *memories*. While the relation between memories and the network's hidden state dynamics was established over the last decade, previous works in this direction were of a predominantly descriptive nature focusing mainly on locating the dynamical objects of interest. In particular, it remained unclear how dynamical observables affect the performance, how they form and whether they can be manipulated. Here, we utilize different training protocols, datasets and architectures to obtain a range of networks solving a delayed classification task with similar performance, alongside substantial differences in their ability to extrapolate for longer delays. We analyze the dynamics of the network's hidden state, and uncover the reasons for this difference. Each memory is found to be associated with a nearly steady state of the dynamics which we refer to as a 'slow point'. Slow point speeds predict extrapolation performance across all datasets, protocols and architectures tested. Furthermore, by tracking the formation of the slow points we are able to understand the origin of differences between training protocols. Finally, we propose a novel regularization technique that is based on the relation between hidden state speeds and memory longevity. Our technique manipulates these speeds, thereby leading to a dramatic improvement in memory robustness over time, and could pave the way for a new class of regularization methods.

1. Introduction

Recurrent Neural Networks (RNN) are the key tool currently used in machine learning when dealing with sequential data (Sutskever et al., 2014), and in many tasks requiring a memory of past events (Oh et al., 2016). This is due to the dependency of the network on its past states, and through them on the entire input history. This ability comes with a cost - RNNs are known to be hard to train (Pascanu et al., 2013a). This difficulty is commonly associated with the vanishing gradient that appears when trying to propagate errors over long times (Hochreiter, 1998). When training is successful, the network's hidden state represents these memories. Understanding how such representation forms throughout training can open new avenues for improving learning of memory-related tasks.

Linking hidden state dynamics with task-related memories requires some form of reverse engineering. This can be done by focusing on individual recurrent units (Karpathy et al., 2015; Oh et al., 2016), or by analyzing global network properties. We opt for the latter, analyzing the RNN's hidden states as a discrete-time dynamical system.

In this framework, memories might be associated with a wide range of dynamical objects. On one extreme, transient dynamics can be harnessed for memory operations (Manjunath & Jaeger, 2013; Maass, 2011; Maass et al., 2002). On the other extreme, there are memory networks (Sukhbaatar et al., 2015) that memorize *everything* and later use only the relevant memories while ignoring all the rest. The idealized dynamical scenario where each memory is associated with a fixed point in the RNN state space (Hopfield, 1982; Sussillo, 2014; Barak, 2017; Amit, 1989) was refined in (Durstewitz, 2003; Sussillo & Barak, 2013; Mante et al., 2013) where points which are only approximately fixed (slow points), with a drift that is slower than the task duration were shown to represent memory.

To allow in-depth analysis of the formation of memories throughout training, we analyze a simple delayed classification task. While simple enough to analyze, the task requires both memory and processing - two key operations in many RNN tasks. We train this task using different

[1] Faculty of Electrical Engineering, Technion, Israel Institute of Technology [2] Network Biology Research Laboratory, Technion, Israel Institute of Technology [3] Rappaport Faculty of Medicine, Technion, Israel Institute of Technology [4] Currently at Weizmann Institute of Science, Israel . Correspondence to: Doron Haviv <Doron.Haviv12@gmail.com>, Alexnader Rivkind <Sashkarivkind@gmail.com>, Omri Barak <Omri.Barak@gmail.com>.

Proceedings of the 36^{th} International Conference on Machine Learning, Long Beach, California, PMLR 97, 2019. Copyright 2019 by the author(s).

Code availabe at: https://github.com/DoronHaviv/MemoryRNN datasets, unit-types and training protocols, to obtain a range of networks. We find that different protocols lead to comparable performance on the task. A more careful analysis, however, reveals that the resulting networks differ in their extrapolation abilities and reflect their training histories.

To uncover the underlying reasons for such differences, we extend tools used in continuous-time systems in neuroscience (Sussillo & Barak, 2013). We find that memories of the different classes are represented by slow points of varying slowness. We show that the speed of the points predicts the extrapolation properties of their associated class. We establish such a correlation in a large variety of settings.

Having established the importance of slow points as a predictor, we obtain an instructive insight on *how* they evolve along the training course. Detailed analysis of individual training trajectories makes it possible to monitor the formation of slow points under a specific training protocol. This technique uncovers an interplay between newly recruited and functional slow points – decreasing the stability of the latter in a systematic manner. This provides a link between training curriculum, dynamical objects, memory and performance.

Ultimately, we take a step from merely predicting performance to improving it. To this extent we modify the loss function to penalize hidden state speed in relevant points, and report a dramatic improvement for extremely long delays.

2. Task Definition

Inspired by real-world applications of Recurrent Neural Networks (Oh et al., 2016), we designed a task where the RNN has to combine stimulus processing and memorization (Figure 1). The network is presented with a series of noisy images, among which appears a single target image (from MNIST or CIFAR-10) at time t_s. At a later time point, t_a, the network receives a response trigger in a separate input channel, prompting it to output the label of the image. At all other times, the network should report a null label.

The stimulus and reporting times are chosen randomly at each trial from a uniform distribution on $[1, T_{max}]$ subject to the constraint $t_a - t_s > 4$. The total stimulation time is $T_{max} = 20$, and the network was requested to distinguish between $|V| = 10$ different classes of MNIST (LeCun et al., 2010) or CIFAR-10 (Krizhevsky et al.).

Each pixel of the noise mask was sampled from a Gaussian distribution with mean and variance matching its counterpart at the image corpus $\epsilon \sim N(\mu_n, \sigma_n^2)$. We tested the RNNs ability to extrapolate from this task to longer durations by increasing the delay T_{max}.

The motivation behind this task is three-fold. First, as explained, this task is comparable to real-world scenarios where RNNs are used, combining the need for both stimulus memorization and feature extraction. Second, the task lends itself to parametric variations, allowing to compare both different training protocols and generalization abilities. Third, desiring to understand the dynamical nature of memorization in discrete Gated-RNNs, the delay between stimulus and response trigger allows for evolution of RNN hidden-state (HS), which can be reliably analyzed using well known methods from dynamical systems (Sussillo & Barak, 2013), which we modify to our discrete setting.

3. Model

For MNIST, the network consists of a single recurrent layer of $d = 200$ gated recurrent units, an output layer of $|V| + 1 = 11$ neurons, $|V| = 10$ neuron for the different classes, and an additional neuron for the null indicator. The input layer has $n + 1$ neurons, where n is the number of pixels in the image and an extra binary input channel for the response trigger $X_r(t)$ defined by:

$$X_r(t) = \begin{cases} 1, & \text{if } t = t_a. \\ 0, & \text{otherwise.} \end{cases} \tag{1}$$

For CIFAR-10, the network was expanded to $d = 400$ recurrent units, along with a convolutional front-end composed of three convolutional layers and two dense layers. To eliminate issues of translational invariance regarding the response trigger and the convolutional front-end, the trigger was added as an extra channel to the final dense layer, right before the recurrent units (Figure 1).

The gated units are either GRU:

$$\begin{aligned} z &= \sigma(W_z I + U_z h_t + b_z) \\ r &= \sigma(W_r I + U_r h_t + b_r) \\ h_{t+1} &= z \circ h_t + (1 - z) \circ \tanh(W_h I + U_h(r \circ h_t) + b_h) \end{aligned} \tag{2}$$

or LSTM:

$$\begin{aligned} f &= \sigma(W_f I + U_f h_t + b_z) \\ i &= \sigma(W_i I + U_i h_t + b_i) \\ o &= \sigma(W_o I + U_o h_t + b_o) \\ c_{t+1} &= f \circ c_t + i \circ \tanh(W_c I + U_c h_t + b_c) \\ h_{t+1} &= o \circ \tanh(c_t) \end{aligned} \tag{3}$$

For the analysis of the network's phase space, we denote the state of the recurrent layer by ξ, which for LSTM is $\xi = \begin{pmatrix} h \\ \tanh(c) \end{pmatrix}$ and for GRU $\xi = h$.

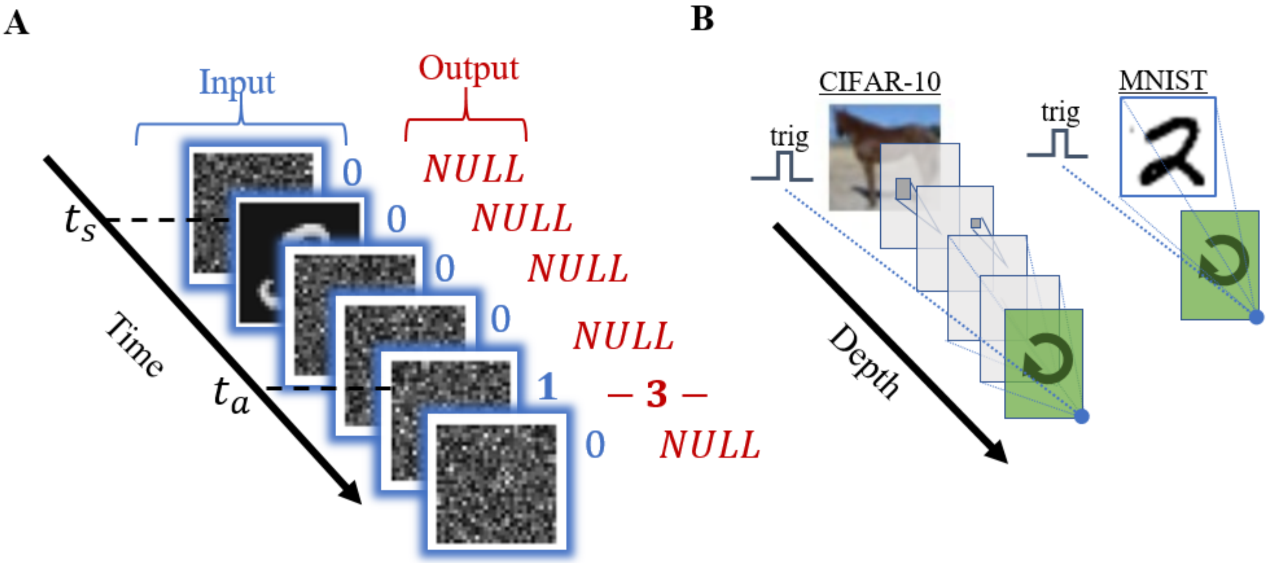

Figure 1. **A** Task. The network is presented an MNIST or CIFAR-10 image amidst noisy images and has to report its label at a later time, as requested by a separate input $(0, 1$ to the right of images). Output should be null at all times except the reporting time. The precise times t_a, t_s vary from trial to trial. **B** Architecture. In the case of MNIST dataset, both the image and the trigger signal are fed directly into the recurrent layer. For the CIFAR-10 task, a convolutional feed forward network is added in front of the recurrent layer, while the trigger signal is connected directly to the RNN.

The network was trained using the 'Adam' optimizer (Kingma & Ba, 2014) with a soft-max cross-entropy loss function with an increased loss on reporting at $t = t_a$ in proportion to T_{max}. Full description of each protocol, including schedules and other hyper-parameters is given in the supplemental code.

4. Training Protocol: Two Types of Curricula

We found that training failed when using straightforward stochastic gradient (SG) optimization on the full task. The network converged to a state where it consistently reports 'null' without regarding neither the output trigger nor the images it has received as inputs. This suboptimal behavior did not improve upon further training. On the other hand, we observed that simpler versions of the task are learn-able. If the maximal delay between stimulus and reporting time was short or when we introduced only a limited number of different digits, the network was able to perform the task. This led us to try two different protocols of curriculum learning (Bengio et al., 2009) in order teach the network the full required task:

1. Vocabulary curriculum (VoCu) - here we started from two classes $V = \{c_1, c_2\}$ and then increased the vocabulary gradually until reaching the full class capacity. This protocol is similar to the original concept of (Bengio et al., 2009) except the fact that in our vocabulary all the classes occur with the same frequency, and the selected order of class introduction is in fact arbitrary.

2. Delay curriculum (DeCu) - starting from short delays between stimulus and reporting times ($T_{max} = 6$), we progressively extended it toward the desired values. Implicitly mentioned in (Hochreiter, 1998), this regime is expected to mitigate the vanishing gradient problem, at least during initial phase of training.

5. Extrapolation Ability Depends on Training Protocol

We found that, in good accordance with existing literature (Bengio et al., 2009; Jozefowicz et al., 2015) results for the nominal test-set were fairly indifferent to the training protocol (Supplementary material). Once we evaluated the ability of each setting to extrapolate to longer delays, however, similarity ends and differences emerge.

We observed how each setting performs when the delay between stimulus and response trigger is extended further beyond $T_{max} = 20$. If $|V| = 10$ robust fixed-point attractors have formed, retrieval accuracy should not be affected by the growing delay. If the computation is based on transients, then all class information is expected to eventually vanish.

Experiments revealed that neither of these extreme options

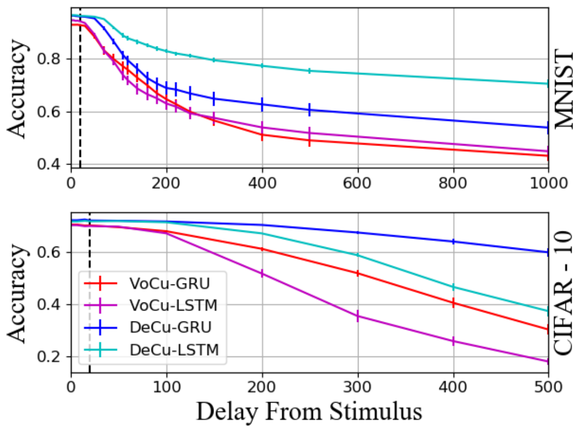

Figure 2. Retrieval accuracy when increasing the delay between stimulus and response trigger beyond $T_{max} = 20$. Despite similar performance initially, the ability to generalize for greater delays than those trained for (dashed vertical lines) varies with protocol. DeCu was superior to Vocu in both LSTM and GRU architectures for both MNIST and CIFAR-10 datasets.

was the case - performance deteriorated with increasing delay, but did not reach chance levels (Figure 2). This deterioration implies that not every memorized digit corresponds to a stable fixed point attractor, but some do. Furthermore, the deterioration was curriculum-dependent, with DeCu outperforming VoCu in all cases.

6. Dynamics of Hidden Representation

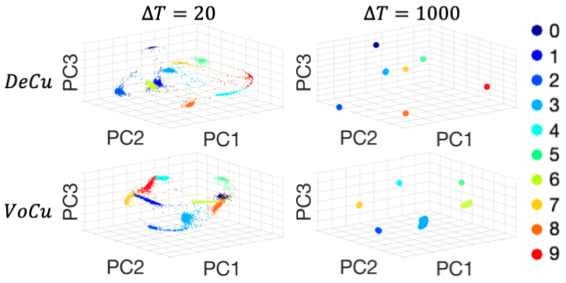

Figure 3. Hidden state projected on leading principal components in GRU - RNN on MNIST for delays of $\Delta t = 20$ and 10^3 time-steps from stimulus. States are color codded by their prediction. For the nominal delay ten distinct regions are observed in the state space, corresponding to each of the $|V| = 10$ classes. Examination of a larger delay $\Delta t = 10^3$ reveals that some clouds collapse into a single point at the center of the cloud, while others vanish completely. The spread of samples in the nominal delay, along with a smaller number of distinct fixed points at $\Delta t = 10^3$ in VoCu aligns with our findings of faster and less stable dynamics compared to DeCu.

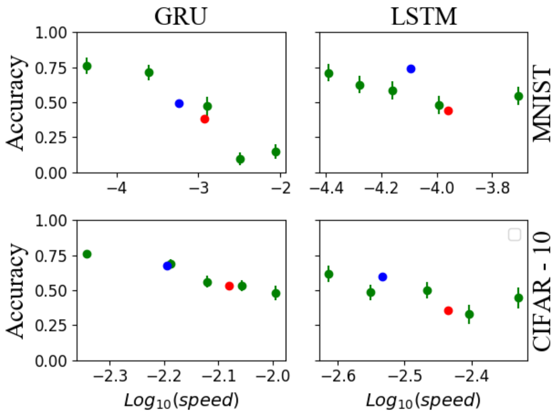

Figure 4. Extrapolation accuracy predicted by slow point speeds. The accuracy at long delays ($\Delta t = 10^3$ for MNIST and $\Delta t = 500$ for CIFAR-10) is shown as a function of the slow point speed of the associated class (green, errorbars denote standard error of the mean). In all datasets, unit-types and training methods, slower speeds correlate with increased accuracy. The red and blue dots show the mean speed and accuracy for each training protocol (standard error of the mean is smaller than marker size). The difference in speeds between the protocols underlies the different extrapolation performance shown in Figure 2. Ten networks were used for MNIST, and five for CIFAR-10.

The relevant phase space of this dynamical system is the recurrent layer state ξ. We thus begin by visually inspecting (in the first 3 PCA components) the activity of the network for the maximal training delay, $\Delta t = 20$. We show here results for the MNIST dataset with a GRU architecture, but similar behavior is seen for other conditions and the statistics of all conditions is analyzed below. The left panels of Figure 3 show that different trials of each digit are well separated into $|V|$ regions with a one to one correspondence to data classes. Following these trajectories for a longer delay of $\Delta t = 1000$ shows that some regions converge into what appears to be fixed points, while others vanish (right panels). These figures also clearly show the difference between the two protocols. While both achieve a good separation with the nominal delay (left), it is already apparent that VoCu leads to clouds of points with a larger spread, possibly indicating a weaker attraction.

To verify the existence or absence of fixed points hinted by the above visualization, we apply an algorithm developed for continuous time vanilla RNN (Sussillo & Barak, 2013) to our setting. Briefly, fixed points (stable or unstable) are local minima of the (scalar) speed $S(\xi, I)$ of the hidden state ξ.

$$S(\xi, I) = ||F(\xi, I) - \xi||_2 \tag{4}$$

where the evolution of state,

$$F(\xi') = \xi(t+1)\Big|_{\xi(t)=\xi'} \tag{5}$$

is given by equations 2 or 3 for GRU or LSTM respectively. It is now possible to use gradient descent on the speed S with respect to state ξ, namely, $\nabla_\xi S$, to locate such minima.

The initial conditions for this gradient descent were obtained by running the network with the mean delay value $\Delta t = 15$, and using the averages of hidden states of each class as initial conditions. The external input I during gradient descent was the average of the noise images, thus effectively making our system Time-Invariant so that such points and their stability are well defined. We verified that using different fixed external inputs did not qualitatively alter the results (not shown). We repeated the procedure for several realizations, and it always resulted in a local minimum of speed for each class (which we call a slow point), with a readout that matches the class label.

To look for a *quantitative* relation between slow point speed and the memory robustness we located slow points for every class, computed their speed and the prediction accuracy of their associated classes after a long delay. Figure 4 shows that the speed of the slow point associated with a certain class can predict how members of that class will react to extrapolation experiments. This trend holds for all architectures, unit types and datasets tested.

The colored dots in Figure 4 denote the mean of the speeds obtained by the two different protocols. The picture here is consistent with that observed in Figure 2, with DeCu outperforming VoCu for all cases. Our results suggest that this difference is mediated by a difference in speeds of the associated slow points.

We also trained networks on an additional task of delayed *matching* (as opposed to classification), and observed the same speed-accuracy anticorrelation. (Supplementary material)

7. Formation of Slow Points - Why Protocols Differ

We saw that the two training protocols lead to a different representation of the stimulus memory by the network, and hence to different dynamical objects. How does training give rise to these differences? To answer this question, we analyze in detail two settings - a GRU and LSTM architectures trained on the MNIST database. We follow the slow points of the velocity backwards in training time to learn how they emerge and change throughout training, and correlate these events with network performance.

We located slow points as described in Section 6 , and then used them as initial conditions for gradient descent on the network defined by the previous training step. We then repeated this procedure iteratively for all training steps. The assumption is that the change in network parameters at each training step will not induce a very large shift in the locations of the relevant slow points, and thus our continuation procedure can track them. This is not clear in the case of VoCu, where one might expect rapid changes whenever a new class is added. Looking at the speeds of the tracked slow points shows that this is indeed the case for VoCu (Figure 5, **A,B**), with all tracked speeds exhibiting such jumps. DeCu, on the other hand, shows a gradual slowing down of the slow points throughout training.

By observing the gradual slowing down in DeCu, it is easy to understand why this protocol improves performance - slow points become slower. But the situation for VoCu is more complicated because introducing new classes qualitatively changes the dynamics. A natural expectation might be that the classes that were presented first will have more time to stabilize, and thus will be the slowest. We saw that this is not the case (not shown), and thus proceeded to look deeper into the class introduction events along VoCu training.

Consider such an event - the introduction of class c_i at training time τ_i (We use τ to denote training step (of SG), as opposed to the time during each trial which is denoted by t). We perform a short backtracking procedure - starting just before the succeeding class is introduced ($\tau_{i+1} - 2000$), and following it back until slightly before it appeared ($\tau_i - 2000$). We verified that we can follow the point despite the jumps mentioned above (shown in supplementary material).

We discovered that the new class is assigned to an existing slow point. This slow point was previously classified as an existing class $j < i$. By stitching together all such backtracking procedures, we obtain a diagram indicating where each new class originated (Figure 5**C**).

Does this history affect performance? To answer this question, we checked the difference in performance of the various classes following the introduction of a new one. For instance, the diagram shows that class 8 originated from class 5. Figure 5**D** shows that the performance of class 5 was impaired more than other existing classes following the introduction of class 8.

To evaluate the statistics of this phenomenon, we repeat this procedure for many networks, and all class introduction events. Figure 5**E** shows that the decline in accuracy of the classes that spawned the new class is significantly larger than that of the other classes (LSTM histogram is shown in supplementary material). Specifically, for class c_i

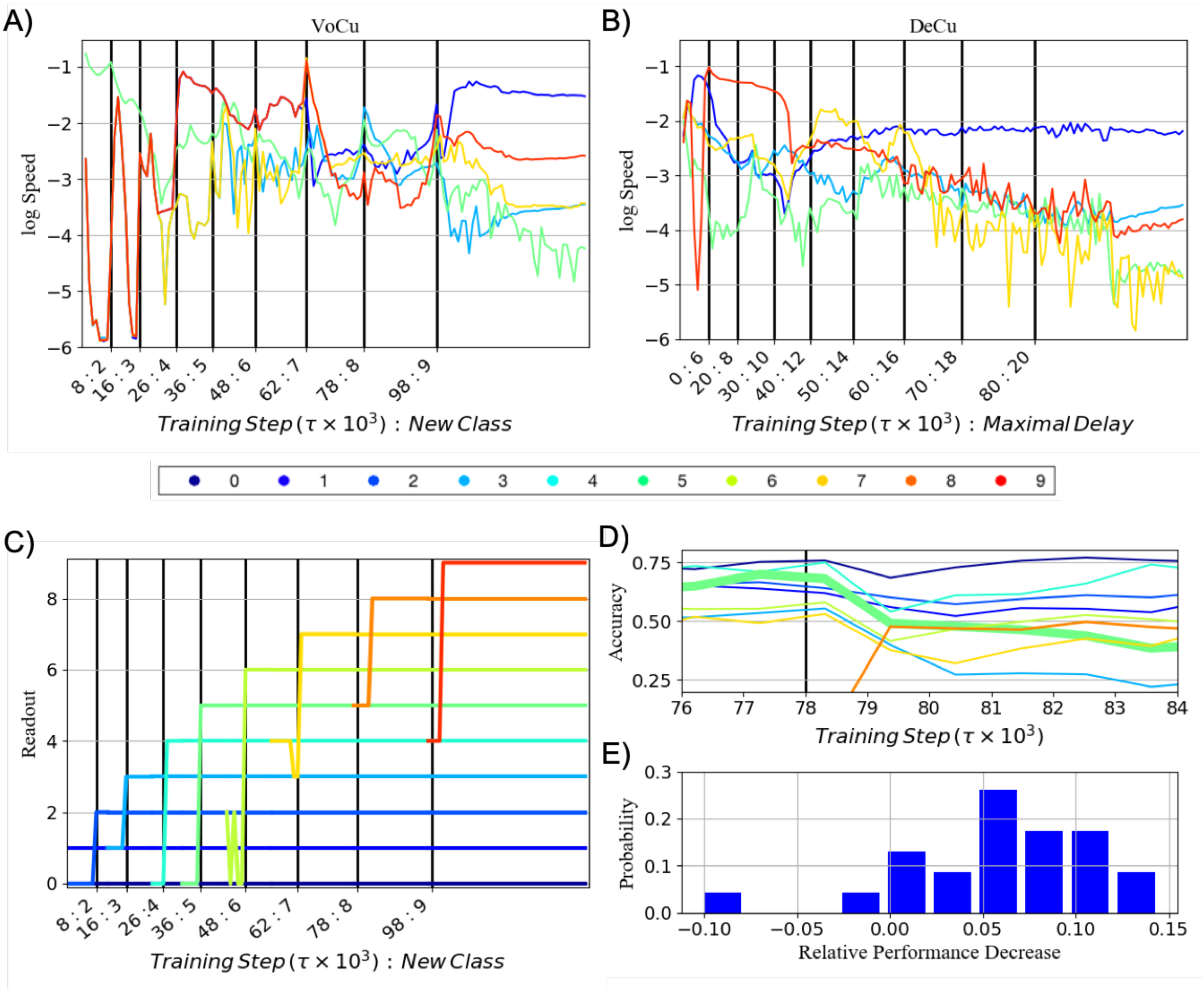

Figure 5. **A,B** Speeds of each slow-point through training (only five are shown for clarity) obtained by iteratively tracking them back in training time. The specific schedule of each curriculum is marked on the time axis ($\tau_i : c_i$ for VoCu and $\tau_i : T_{max,i}$ for Decu). VoCu shows sharp jumps in the speed of all points for each class introduction, in contrast to DeCu which exhibits a gradual slowing down along the whole course of training. **C** A branching diagram for VoCu, obtained from short backtracking every class, proximal to its appearance. The change in readout suggests which slow point gave rise to the new point (e.g., 8 from 5 at time 78; 7 from 3 or 4 at time 62). **D** The accuracies of each individual class for the same VoCu realization when class '8' was introduced. As a result, all previously existing classes show a degradation in accuracy, however, the class from which class '8' emerged from (class '5') exhibited a stronger decline. Noisy images with double amplitude ($2\sigma_n^2$) were used to amplify the effect strength. **E** Verifying the statistical significance of the result in (D). For each branching event, we compared the accuracy drop of classes that gave rise to the new class, with the drop for the remaining classes. The histogram is from all events in three VoCu realizations. It shows that indeed accuracy of spawning classes decreases more following a branching event.

being newly introduced at training step τ_i, accuracy of all classes $\{c_j\}_{j<i}$ is evaluated at training steps $\tau = \tau_i - 1000$ and $\tau = \tau_i + 4000$. Accuracy change Δa_k for class c_k that branches into class i is compared to the average $\langle \Delta a_j \rangle_{j<i, j \neq k}$.

8. Improving Long Term Memory

Can the aforementioned insights help improve performance? Can we obtain memory for delays that are substantially longer than in the scenarios used for training? To answer this question, we first trained the network as before using one of the protocols discussed above. We then trained for an additional period of 5000 gradient steps (compared to 140000 in initial training) while regularizing our standard cross-entropy loss function L_{xent} by a term accounting for hidden state speed. The new loss:

$$L' = L_{xent} + \lambda \sum_{i \in V} |S(\bar{\xi_i})| \tag{6}$$

penalizes for high speed S of equation 4 at representative points $\bar{\xi_i}$ associated with each class.

The natural candidates for $\bar{\xi_i}$ are the slow points discussed above, and indeed Figure 6 shows that dramatic improvements were achieved, when compared to the same training without the added regularization. Using the slow points as regularizer targets is still somewhat costly, as their detection requires a gradient descent step. Furthermore, as training proceeds, the location of slow points can move – rendering the original regularization targets less effective. We thus used a proxy for the slow points by taking $\bar{\xi_i}$ to be the centre of mass of each class. This simpler procedure achieved results that were comparable to using the slow points themselves. In both cases, we verified that slow point speed was manipulated to achieve the improved performance (Figure 6 legend), and that the accuracy for nominal delays remained virtually intact. In principle, one could train for such long delays by straight forward back propagation through time, but this would be orders of magnitude more time consuming than our method, if not impossible.

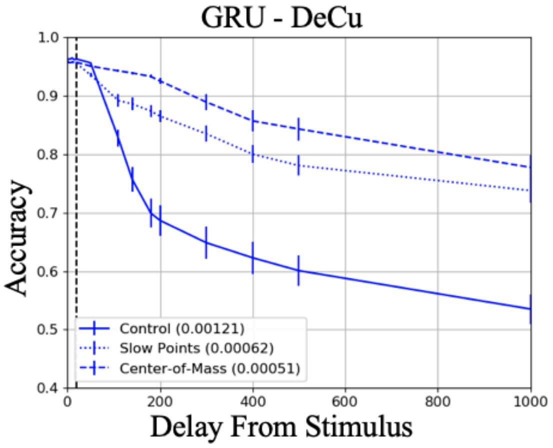

Figure 6. Effect of speed regularization on the performance is demonstrated for the DeCu training on GRU architechure on MNIST, results for all other setting are in Supplementary material. For control, we trained the network for the same number of additional gradient decent steps without any regularization (solid). Regularization targets were either the slow-points (dotted) or the center-of-mass (dashed). Both regularization methods resulted in dramatic improvements compared to control, which is also reflected in a smaller speed of slow-points after additional training (shown in legend brackets).

9. Discussion

Training RNNs to perform memory-related tasks is difficult (Pascanu et al., 2013b), and as a consequence many suggestions were made on how to alleviate this difficulty. Changing network architecture, unit-types or training protocols might be expected to generate different solutions to the same task.

Here we showed that different training protocols can lead to different locally optimal solutions. Although these solutions perform similarly under nominal conditions, challenging the networks with unforeseen settings reveals their differences.

An RNN is a dynamical system, and as such its operation can be understood in the language of fixed points and other dynamical objects. By analyzing the phase space of the network's hidden states, we showed that the memory of each class was associated with a slow point of the dynamics. Such slow points were shown to assist network functionality (Machens et al., 2005), and arise through training (Mante et al., 2013; Durstewitz, 2003). The speeds of these slow points were highly correlated to the functional characteristics of memory longevity, and thus provide a dynamical explanation of the idiosyncrasies observed between curricula and architectures. Our result proved valid across architectures, datasets and unit types.

In specific cases, we were able to follow the formation of the recruitment of slow points to representation of memories during the training process. Such recruited slow points reside in an area of phase space that *belongs* to a specific existing class. We showed that this process is associated with a decrease in network accuracy that is specific for the classes that contribute the slow point. Things could have been different - slow points could emerge in an area of phase space that is distant from existing ones, and the introduction of a new class could have resulted in a uniform effect on all existing classes. To uncover this process, we introduced a back-tracking methodology that could be relevant to any case in which learning modifies the dynamics of the network. Possible applications include studying the success and failure in creating memories (Beer & Barak, 2018), preventing catastrophic forgetting, understanding memory capacity and more.

The setting studied in this work is a particular case of the more general problem of solution equivalence in gradient based optimization: On the one hand theoretical and numerical evidence does exist for training outcome's indifference to protocol details. Specifically to our case of RNN, it is shown in (Cirik et al., 2016) that, at least for language modeling tasks, performance does not heavily depend on training protocol. On the other hand, such an indifference is far from being fully established. In particular, stochastic gradient optimization suffers from known drawbacks (Dauphin et al., 2014; Martens & Sutskever, 2011) and might prove dependent on initialization (Sutskever et al., 2013) (and in particular pre-training). Our results show that networks with the same initialization can reach different solutions, and begin to uncover the dynamics underlying the route to these solutions.

Ultimately, at the engineering end of this study, the aforementioned insights allowed us to come up with a novel regularization technique which, by penalizing the hidden state speeds, enables a dramatic improvement of performance for extremely long times, while keeping nominal performance intact. Having noticed that slowly increasing the delay (DeCu) resulted in better stability and increased extrapolation ability for all the conditions tested, it might have been possible to reach similar performance by extending the DeCu protocol to very large delays. Such long backpropagation through time, however, is extremely costly, if not impossible, and avoided by our new method. Notably, our method preserves linear time complexity of plain RNN, as opposed to attention mechanisms whose complexity is quadratic in time (Ke et al. (2018) and references therein).

Importantly, our regularization procedure relies on an informed guess of *what* information the system needs to memorize to improve performance and of *where* such information may be stored. In a delayed classification task, the trivial and straight forward guess is that remembering *class* improves performance with relevant information being represented by the mean of class' samples hidden state $\bar{\xi}_i$. Future work may generalize our methodology to more complex tasks, such as natural language processing, where long term memorization amid a continuous stream of potentially useful input remains an open challenge (Paperno et al., 2016). Doing so will require identifying the appropriate entities to memorize and the locations in hidden space representing these memories (Bau et al., 2018), for use as regularization targets.

Acknowledgements

OB is supported by Israel Science Foundation (346/16). The Titan Xp used for this research was donated by the NVIDIA Corporation.

References

Amit, D. J. Modeling brain function: the world of attractor neural networks, 1st edition. 1989.

Barak, O. Recurrent neural networks as versatile tools of neuroscience research. *Current opinion in neurobiology*, 46:1–6, 2017.

Bau, A., Belinkov, Y., Sajjad, H., Durrani, N., Dalvi, F., and Glass, J. Identifying and controlling important neurons in neural machine translation. *arXiv preprint arXiv:1811.01157*, 2018.

Beer, C. and Barak, O. One step back, two steps forward: interference and learning in recurrent neural networks. *arXiv preprint arXiv:1805.09603*, 2018.

Bengio, Y., Louradour, J., Collobert, R., and Weston, J. Curriculum learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pp. 41–48, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-516-1. doi: 10.1145/1553374.1553380. URL http://doi.acm.org/10.1145/1553374.1553380.

Cirik, V., Hovy, E. H., and Morency, L. Visualizing and understanding curriculum learning for long short-term memory networks. *CoRR*, abs/1611.06204, 2016. URL http://arxiv.org/abs/1611.06204.

Dauphin, Y. N., Pascanu, R., Gulcehre, C., Cho, K., Ganguli, S., and Bengio, Y. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. In *Advances in neural information processing systems*, pp. 2933–2941, 2014.

Durstewitz, D. Self-organizing neural integrator predicts interval times through climbing activity. *Journal of Neuroscience*, 23(12):5342–5353, 2003.

Hochreiter, S. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6(02):107–116, 1998.

Hopfield, J. J. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the national academy of sciences*, 79(8):2554–2558, 1982.

Jozefowicz, R., Zaremba, W., and Sutskever, I. An empirical exploration of recurrent network architectures. In *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, pp. 2342–2350. JMLR.org, 2015. URL http://dl.acm.org/citation.cfm?id=3045118.3045367.

Karpathy, A., Johnson, J., and Fei-Fei, L. Visualizing and understanding recurrent networks. *arXiv preprint arXiv:1506.02078*, 2015.

Ke, N. R., GOYAL, A. G. A. P., Bilaniuk, O., Binas, J., Mozer, M. C., Pal, C., and Bengio, Y. Sparse attentive backtracking: Temporal credit assignment through reminding. In *Advances in Neural Information Processing Systems*, pp. 7640–7651, 2018.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Krizhevsky, A., Nair, V., and Hinton, G. Cifar-10 (canadian institute for advanced research). URL http://www.cs.toronto.edu/~kriz/cifar.html.

LeCun, Y., Cortes, C., and Burges, C. Mnist handwritten digit database. *AT&T Labs [Online]. Available: http://yann. lecun. com/exdb/mnist*, 2, 2010.

Maass, W. Liquid state machines: motivation, theory, and applications. In *Computability in context: computation and logic in the real world*, pp. 275–296. World Scientific, 2011.

Maass, W., Natschläger, T., and Markram, H. Real-time computing without stable states: A new framework for neural computation based on perturbations. *Neural computation*, 14(11):2531–2560, 2002.

Machens, C. K., Romo, R., and Brody, C. D. Flexible control of mutual inhibition: a neural model of two-interval discrimination. *Science*, 307(5712):1121–1124, 2005.

Manjunath, G. and Jaeger, H. Echo state property linked to an input: Exploring a fundamental characteristic of recurrent neural networks. *Neural Computation*, 25(3):671–696, 2013. doi: 10.1162/NECO_a_00411. URL https://doi.org/10.1162/NECO_a_00411. PMID: 23272918.

Mante, V., Sussillo, D., Shenoy, K. V., and Newsome, W. T. Context-dependent computation by recurrent dynamics in prefrontal cortex. In *Nature*, 2013.

Martens, J. and Sutskever, I. Learning recurrent neural networks with hessian-free optimization. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pp. 1033–1040. Citeseer, 2011.

Oh, J., Chockalingam, V., Singh, S. P., and Lee, H. Control of memory, active perception, and action in minecraft. *CoRR*, abs/1605.09128, 2016. URL http://arxiv.org/abs/1605.09128.

Paperno, D., Kruszewski, G., Lazaridou, A., Pham, Q. N., Bernardi, R., Pezzelle, S., Baroni, M., Boleda, G., and Fernández, R. The lambada dataset: Word prediction requiring a broad discourse context. *arXiv preprint arXiv:1606.06031*, 2016.

Pascanu, R., Mikolov, T., and Bengio, Y. On the difficulty of training recurrent neural networks. In *International Conference on Machine Learning*, pp. 1310–1318, 2013a.

Pascanu, R., Mikolov, T., and Bengio, Y. On the difficulty of training recurrent neural networks. In *International Conference on Machine Learning*, pp. 1310–1318, 2013b.

Sukhbaatar, S., Weston, J., Fergus, R., et al. End-to-end memory networks. In *Advances in neural information processing systems*, pp. 2440–2448, 2015.

Sussillo, D. Neural circuits as computational dynamical systems. *Current opinion in neurobiology*, 25:156–63, 2014.

Sussillo, D. and Barak, O. Opening the black box: Low-dimensional dynamics in high-dimensional recurrent neural networks. *Neural Computation*, 25(3):626–649, 2013. doi: 10.1162/NECO_a_00409. URL https://doi.org/10.1162/NECO_a_00409. PMID: 23272922.

Sutskever, I., Martens, J., Dahl, G., and Hinton, G. On the importance of initialization and momentum in deep learning. In *International conference on machine learning*, pp. 1139–1147, 2013.

Sutskever, I., Vinyals, O., and Le, Q. V. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pp. 3104–3112, 2014.