HLD 2023: 1st Workshop on High-dimensional Learning Dynamics

Elephant Neural Networks: Born to Be a Continual Learner

Qingfeng Lan QLAN3@UALBERTA.CA
University of Alberta, Edmonton, Canada

A. Rupam Mahmood ARMAHMOOD@UALBERTA.CA
University of Alberta, Edmonton, Canada

Abstract

Catastrophic forgetting remains a great challenge to continual learning for decades. While recent works have proposed effective methods to mitigate this problem, they mainly focus on the algorithmic side. Meanwhile, we do not fully understand what architectural properties of neural networks lead to catastrophic forgetting. This study aims to fill this gap by studying the role of activation functions in the training dynamics of neural networks and their impact on catastrophic forgetting. Our study reveals that, besides sparse representations, the gradient sparsity of activation functions also plays an important role in reducing forgetting. Based on this insight, we propose a new class of activation functions, elephant activation functions, that can generate both sparse representations and sparse gradients. We show that the resilience of neural networks to forgetting can be significantly improved by simply replacing classical activation functions with elephant activation functions.

1. Introduction

One of the biggest challenges to continual learning is the issue of *catastrophic forgetting* [18]. Catastrophic forgetting stands for the phenomenon that neural networks tend to forget prior knowledge drastically when learned with stochastic gradient descent algorithms on non-independent and identically distributed (non-iid) data. Many effective methods have been proposed to reduce forgetting, such as replay [39], regularization [46], and optimization-based methods [13]. However, instead of alleviating forgetting by designing neural networks with specific properties, most of these methods circumvent the problem by focusing on the algorithmic side. *There is still a lack of full understanding of what properties of neural networks lead to catastrophic forgetting.*

Recently, Mirzadeh et al. [40, 41] studied the forgetting problem with various neural network architectures, demonstrating that architectures can play a role that is as important as algorithms in continual learning. *Yet the interactions between continual learning and neural network architectures remain to be under-explored.* In this work, we aim to better understand and reduce forgetting by studying the impact of different neural network architectural choices on continual learning. Specifically, we focus on activation functions and investigate the role of activation functions in the training dynamics of neural networks. Our analysis suggests that not only sparse representations but also sparse gradients are essential for continual learning. Based on this discovery, we design a new class of activation functions, elephant activation functions, which are able to generate both sparse function values and sparse gradients. We show that by incorporating elephant activation functions, neural networks are more resilient to catastrophic forgetting in streaming learning for regression.

2. Investigating catastrophic forgetting via training dynamics

Firstly, we look into the forgetting issue via the training dynamics of neural networks. Consider a simple regression task. Let a scalar function $f_{\boldsymbol{w}}(\boldsymbol{x})$ be represented as a neural network, parameterized by $\boldsymbol{w}$, with input $\boldsymbol{x}$. $F(\boldsymbol{x})$ is the true function and the loss function is $L(f, F, \boldsymbol{x})$. For example, for squared error, we have $L(f, F, \boldsymbol{x}) = (f_{\boldsymbol{w}}(\boldsymbol{x}) - F(\boldsymbol{x}))^2$. At each time step t, a new sample $\{\boldsymbol{x}_t, F(\boldsymbol{x}_t)\}$ arrives. Given this new sample, to minimize the loss function $L(f, F, \boldsymbol{x}_t)$, we update the weight vector by $\boldsymbol{w}' = \boldsymbol{w} + \Delta_{\boldsymbol{w}}$ where $\Delta_{\boldsymbol{w}}$ is the weight difference. With the stochastic gradient descent (SGD) algorithm, we have $\boldsymbol{w}' = \boldsymbol{w} - \alpha \nabla_{\boldsymbol{w}} L(f, F, \boldsymbol{x}_t)$, where α is the learning rate. So $\Delta_{\boldsymbol{w}} = \boldsymbol{w}' - \boldsymbol{w} = -\alpha \nabla_{\boldsymbol{w}} L(f, F, \boldsymbol{x}_t) = -\alpha \nabla_f L(f, F, \boldsymbol{x}_t) \nabla_{\boldsymbol{w}} f_{\boldsymbol{w}}(\boldsymbol{x}_t)$. By Taylor expansion, we have $f_{\boldsymbol{w}'}(\boldsymbol{x}) - f_{\boldsymbol{w}}(\boldsymbol{x}) = -\alpha \nabla_f L(f, F, \boldsymbol{x}_t) \langle \nabla_{\boldsymbol{w}} f_{\boldsymbol{w}}(\boldsymbol{x}), \nabla_{\boldsymbol{w}} f_{\boldsymbol{w}}(\boldsymbol{x}_t) \rangle + O(\Delta_{\boldsymbol{w}}^2)$, where $\langle \cdot, \cdot \rangle$ denotes Frobenius inner product or dot product depending on the context. In this equation, since $-\alpha \nabla_f L(f, F, \boldsymbol{x}_t)$ is unrelated to $\boldsymbol{x}$, we only consider the quantity $\langle \nabla_{\boldsymbol{w}} f_{\boldsymbol{w}}(\boldsymbol{x}), \nabla_{\boldsymbol{w}} f_{\boldsymbol{w}}(\boldsymbol{x}_t) \rangle$, which is known as the neural tangent kernel (NTK) [23]. Without loss of generality, assume that the original prediction $f_{\boldsymbol{w}}(\boldsymbol{x}_t)$ is wrong, i.e. $f_{\boldsymbol{w}}(\boldsymbol{x}_t) \neq F(\boldsymbol{x}_t)$ and $\nabla_f L(f, F, \boldsymbol{x}_t) \neq 0$. To correct the wrong prediction while avoiding forgetting, we expect this NTK to satisfy two properties that are essential for continual learning:

Property 1 (error correction) *For* $\boldsymbol{x} = \boldsymbol{x}_t$, $\langle \nabla_{\boldsymbol{w}} f_{\boldsymbol{w}}(\boldsymbol{x}), \nabla_{\boldsymbol{w}} f_{\boldsymbol{w}}(\boldsymbol{x}_t) \rangle \neq 0$.

Property 2 (zero forgetting) *For* $\boldsymbol{x} \neq \boldsymbol{x}_t$, $\langle \nabla_{\boldsymbol{w}} f_{\boldsymbol{w}}(\boldsymbol{x}), \nabla_{\boldsymbol{w}} f_{\boldsymbol{w}}(\boldsymbol{x}_t) \rangle = 0$.

In particular, Property 1 allows for error correction by optimizing $f_{\boldsymbol{w}'}(\boldsymbol{x}_t)$ towards the true value $F(\boldsymbol{x}_t)$, so that we can learn new knowledge (i.e., update the learned function). Essentially, Property 1 requires the gradient norm to be non-zero. On the other hand, Property 2 is much harder to be satisfied. Essentially, to make this property hold, except for $\boldsymbol{x} = \boldsymbol{x}_t$, f is required to achieve zero knowledge forgetting after one step optimization, i.e. $\forall \boldsymbol{x} \neq \boldsymbol{x}_t$, $f_{\boldsymbol{w}'}(\boldsymbol{x}) = f_{\boldsymbol{w}}(\boldsymbol{x})$. *It is the violation of Property 2 that leads to the forgetting issue.* For tabular cases (e.g., $\boldsymbol{x}$ is a one-hot vector and $f_{\boldsymbol{w}}(\boldsymbol{x})$ is a linear function), this property may hold by sacrificing the generalization ability of deep neural networks. In order to benefit from generalization, we introduce Property 3 by relaxing Property 2.

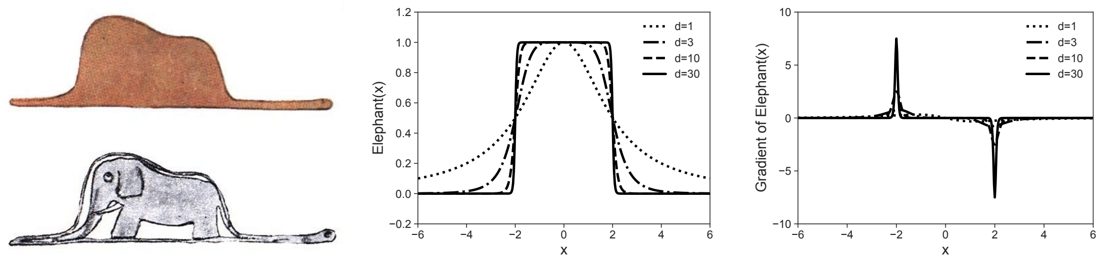
Property 3 (local elasticity) $\langle \nabla_{\boldsymbol{w}} f_{\boldsymbol{w}}(\boldsymbol{x}), \nabla_{\boldsymbol{w}} f_{\boldsymbol{w}}(\boldsymbol{x}_t) \rangle \approx 0$ *for* $\boldsymbol{x}$ *that is dissimilar to* $\boldsymbol{x}_t$ *in a certain sense.*

Property 3 is known as local elasticity [21]. A function f is locally elastic if $f_{\boldsymbol{w}}(\boldsymbol{x})$ is not significantly changed, after it is updated at $\boldsymbol{x}_t$ that is dissimilar to $\boldsymbol{x}$ in a certain sense. For example, we can characterize the dissimilarity with the 2-norm distance.

3. Understanding the success and failure of sparse representations

The above properties can help to deepen our understanding of the success and failure of sparse representations in continual learning. Specifically, we argue that sparse representations are effective to reduce forgetting in linear approximations, but are less useful in non-linear approximations.

Formally, let $\boldsymbol{x}$ be an input and ϕ be an encoder that transforms the input $\boldsymbol{x}$ into its representation $\phi(\boldsymbol{x})$. The representation $\phi(\boldsymbol{x})$ is sparse when most of its elements are zeros. First, consider the case of linear approximations. A linear function is defined as $f_{\boldsymbol{w}}(\boldsymbol{x}) = \boldsymbol{w}^\top \phi(\boldsymbol{x}) : \mathbb{R}^n \mapsto \mathbb{R}$, where



(*a*) A drawing of an elephant. (*b*) $\text{Elephant}(x)$ (*c*) $\text{Elephant}'(x)$

Figure 1: (a) "My drawing was not a picture of a hat. It was a picture of a boa constrictor digesting an elephant." *The Little Prince*, by Antoine de Saint Exupéry. (b) Elephant functions with $a = 2$ and various d. (c) The gradient of elephant functions with $a = 2$ and various d.

$\boldsymbol{x} \in \mathbb{R}^n$ is an input, $\phi : \mathbb{R}^n \mapsto \mathbb{R}^m$ is a fixed encoder, and $\boldsymbol{w} \in \mathbb{R}^m$ is a weight vector. Assume that $\phi(\boldsymbol{x})$ is sparse and non-zero (i.e., $\|\phi(\boldsymbol{x})\|_2 > 0$) for $\boldsymbol{x} \in \mathbb{R}^n$. Easy to know that $\nabla_{\boldsymbol{w}} f_{\boldsymbol{w}}(\boldsymbol{x}) = \phi(\boldsymbol{x})$. We then have $f_{\boldsymbol{w}'}(\boldsymbol{x}) - f_{\boldsymbol{w}}(\boldsymbol{x}) = -\alpha \nabla_f L(f, F, \boldsymbol{x}_t)\, \phi(\boldsymbol{x})^\top \phi(\boldsymbol{x}_t)$. By assumption, Property 1 holds since $f_{\boldsymbol{w}'}(\boldsymbol{x}_t) - f_{\boldsymbol{w}}(\boldsymbol{x}_t) = -\alpha \nabla_f L(f, F, \boldsymbol{x}_t)\, \|\phi(\boldsymbol{x}_t)\|_2^2 \neq 0$. Moreover, when $\boldsymbol{x} \neq \boldsymbol{x}_t$, it is very likely that $\langle \phi(\boldsymbol{x}), \phi(\boldsymbol{x}_t) \rangle \approx 0$ due to the sparsity of $\phi(\boldsymbol{x})$ and $\phi(\boldsymbol{x}_t)$. Thus, $f_{\boldsymbol{w}'}(\boldsymbol{x}) - f_{\boldsymbol{w}}(\boldsymbol{x}) = -\alpha \nabla_f L(f, F, \boldsymbol{x}_t)\, \langle \phi(\boldsymbol{x}), \phi(\boldsymbol{x}_t) \rangle \approx 0$ and Property 3 holds. By satisfying Property 1 and Property 3, sparse representations help mitigate catastrophic forgetting in linear approximations. However, for non-linear approximations, sparse representations can no longer guarantee Property 3. Consider an MLP with one hidden layer. We compute the NTK in this case, resulting in the following lemma.

Lemma 4 (NTK in non-linear approximations) *Given a non-linear function* $f_{\boldsymbol{w}}(\boldsymbol{x}) = \boldsymbol{u}^\top \phi_{\boldsymbol{\theta}}(\boldsymbol{x}) : \mathbb{R}^n \mapsto \mathbb{R}$, *where* $\phi_{\boldsymbol{\theta}}(\boldsymbol{x}) = \sigma(\mathbf{V}\boldsymbol{x} + \boldsymbol{b})$, $\boldsymbol{\theta} = \{\mathbf{V}, \boldsymbol{b}\}$, σ *is a non-linear activation function,* $\boldsymbol{x} \in \mathbb{R}^n$, $\boldsymbol{u} \in \mathbb{R}^m$, $\mathbf{V} \in \mathbb{R}^{m \times n}$, $\boldsymbol{b} \in \mathbb{R}^m$, *and* $\boldsymbol{w} = \{\boldsymbol{u}, \mathbf{V}, \boldsymbol{b}\}$. *The NTK of this non-linear function is* $\langle \nabla_{\boldsymbol{w}} f_{\boldsymbol{w}}(\boldsymbol{x}), \nabla_{\boldsymbol{w}} f_{\boldsymbol{w}}(\boldsymbol{x}_t) \rangle = \phi_{\boldsymbol{\theta}}(\boldsymbol{x})^\top \phi_{\boldsymbol{\theta}}(\boldsymbol{x}_t) + \boldsymbol{u}^\top \boldsymbol{u} (\boldsymbol{x}^\top \boldsymbol{x}_t + 1) \phi'_{\boldsymbol{\theta}}(\boldsymbol{x})^\top \phi'_{\boldsymbol{\theta}}(\boldsymbol{x}_t)$, *where* $\langle \cdot, \cdot \rangle$ *denotes Frobenius inner product or dot product depending on the context.*

The proof of this lemma can be found in the appendix. Compared with the NTK in linear approximations, the equation in Lemma 4 has an additional term $\boldsymbol{u}^\top \boldsymbol{u} (\boldsymbol{x}^\top \boldsymbol{x}_t + 1) \phi'_{\boldsymbol{\theta}}(\boldsymbol{x})^\top \phi'_{\boldsymbol{\theta}}(\boldsymbol{x}_t)$, due to a learnable encoder $\phi_{\boldsymbol{\theta}}$. With sparse representations, we have $\phi_{\boldsymbol{\theta}}(\boldsymbol{x})^\top \phi_{\boldsymbol{\theta}}(\boldsymbol{x}_t) \approx 0$. However, it is not necessary true that $\boldsymbol{u}^\top \boldsymbol{u} (\boldsymbol{x}^\top \boldsymbol{x}_t + 1) \phi'_{\boldsymbol{\theta}}(\boldsymbol{x})^\top \phi'_{\boldsymbol{\theta}}(\boldsymbol{x}_t) \approx 0$ even when $\boldsymbol{x}$ and $\boldsymbol{x}_t$ are quite dissimilar, which violates Property 3. To conclude, our analysis indicates that sparse representations alone are not effective enough to reduce forgetting in non-linear approximations.

4. Obtaining sparsity with elephant activation functions

Although Lemma 4 shows that the forgetting issue can not be fully addressed with sparse representations solely, it also points out a possible solution: sparse gradients. With sparse gradients, we could have $\phi'_{\boldsymbol{\theta}}(\boldsymbol{x})^\top \phi'_{\boldsymbol{\theta}}(\boldsymbol{x}_t) \approx 0$. Together with sparse representations, we may still satisfy Property 3 in non-linear approximations, and thus reduce more forgetting. Specifically, to obtain both sparse representations and sparse gradients, we propose a novel class of bell-shaped activation functions, elephant activation functions [1]. Formally, an elephant function is defined as $\text{Elephant}(x) = \frac{1}{1+(\frac{x}{a})^{2d}}$,

1. We name this bell-shaped activation function as the *elephant* function since the bell-shape is similar to the shape of an elephant (see Figure 1), honoring the work *The Little Prince* by Antoine de Saint Exupéry. This name also hints that

where a controls the width of the function and d controls the slope. We call a neural network that uses elephant activation functions an *elephant neural network (ENN)*. For example, for MLPs and CNNs, we have *elephant MLPs (EMLPs)* and *elephant CNNs (ECNNs)*, respectively.

As shown in Figure 1, elephant activation functions have both sparse function values and sparse gradient values. Specifically, d controls the sparsity of gradients for elephant functions. The larger the value of d, the sharper the slope of an elephant function and the sparser the gradient. On the other hand, a stands for the width of the function, controlling the sparsity of the function itself.

Next, we show that Property 3 holds under certain conditions for elephant functions.

Theorem 5 *Define $f_{\boldsymbol{w}}(\boldsymbol{x})$ as in Theorem 4. Let σ be the elephant activation function with $d \to \infty$. When $|\mathbf{V}(\boldsymbol{x} - \boldsymbol{x}_t)| \succ 2a\mathbf{1}_m$, we have $\langle \nabla_{\boldsymbol{w}} f_{\boldsymbol{w}}(\boldsymbol{x}), \nabla_{\boldsymbol{w}} f_{\boldsymbol{w}}(\boldsymbol{x}_t) \rangle = 0$, where $\succ$ denotes an element-wise inequality symbol and $\mathbf{1}_m = [1, \cdots, 1]^\top \in \mathbb{R}^m$.*

Theorem 5 mainly proves that when $d \to \infty$, Property 3 holds for an EMLP with one hidden layer. However, even when d is small, EMLPs still exhibit local elasticity, as we will show next.

5. Experiment

In this section, we will show that (1) sparse representations alone are not enough to address the forgetting issue, (2) ENNs can continually learn to solve regression tasks by reducing forgetting, and (3) ENNs are locally elastic even when d is a small integer.

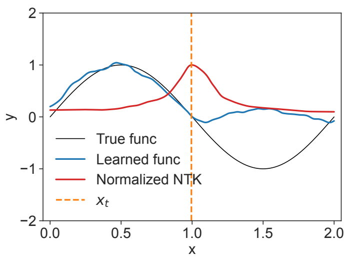
We perform experiments in a simple regression task in the streaming learning setting. In this setting, a learning agent is presented with one sample only at each time step and then performs learning updates given this new sample. Moreover, the learning process happens in a single pass of the whole dataset. Furthermore, the data stream is assumed to be non-iid. Finally, the evaluation happens after each new sample arrives, which requires the agent to learn quickly while avoiding catastrophic forgetting. The task is to approximate a sine function $y = \sin(\pi x)$ where non-iid samples arrive in a stream. The learning agent is an MLP with one hidden layer of size $1,000$, optimized by minimizing the square error loss. During an evaluation, the agent performance is measured by the mean square error (MSE) on a test dataset. We compare our method (EMLP) with two kinds of baselines. One is an MLP with classical activation functions, including ReLU, $\mathrm{Sigmoid}$, ELU, and Tanh. The other is the sparse representation neural network (SR-NN) [34] which is designed to generate sparse representations. We set $d = 4$ for all elephant functions applied in EMLP. Additional training details are included in the appendix.

The test MSEs are summarized in Table 2 in the appendix. EMLP achieves the best performance, reaching a very low test MSE (0.0081) compared with baselines. SR-NN achieves slightly better performance than MLPs with classical activation functions, i.e. 0.4061 v.s. $0.44 \sim 0.47$. However, its test MSE is still large, indicating that the SR-NN fails to approximate $\sin(\pi x)$.

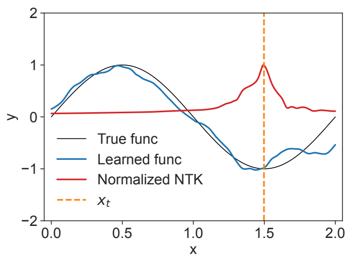
Next, we plot the true function $\sin(\pi x)$, the learned function $f(x)$, and the NTK function $\mathrm{NTK}(x) = \langle \nabla_{\boldsymbol{w}} f_{\boldsymbol{w}}(x), \nabla_{\boldsymbol{w}} f_{\boldsymbol{w}}(x_t) \rangle$ at different training stages for EMLP in Figure 2. It shows that for EMLP with $d = 4$, $\mathrm{NTK}(x)$ quickly decreases to 0 as x moving away from x_t, demonstrating the local elasticity of EMLP with a small d.

By injecting local elasticity to a neural network, we can break the inherent global generalization ability of the neural networks [19], constraining the output changes of the neural network to small

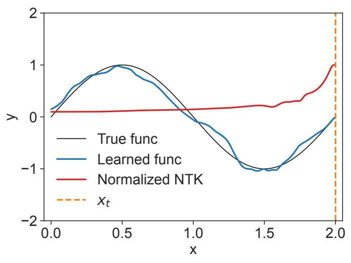
this activation function empowers neural networks with continual learning ability, echoing the old saying that "an elephant never forgets".



(*a*) EMLP at $t = 100$

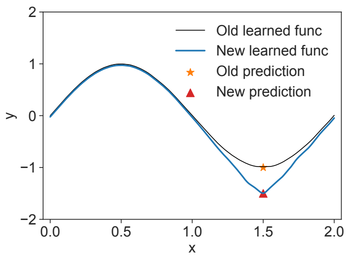


(*b*) EMLP at $t = 150$

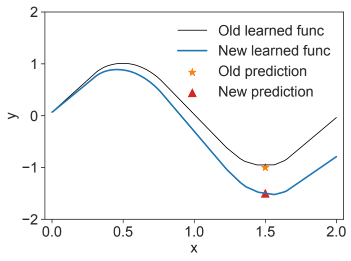


(*c*) EMLP at $t = 200$

Figure 2: Plots of the true function $\sin(\pi x)$, the learned function $f(x)$, and the NTK function $\mathrm{NTK}(x)$ at different training stages for EMLP.



(*a*) EMLP (Elephant)



(*b*) MLP (ReLU)

Figure 3: A demonstration of local elasticity of the EMLP.

local areas. Utilizing this phenomenon, we can update a wrong prediction by "editing" outputs of a neural network nearly point-wisely. To verify, we first train a neural network to approximate $\sin(\pi x)$ well in the traditional supervised learning style. We call the function the old learned function at this stage. Now assume that the original y value (-1.0) for input $x = 1.5$ is changed to $y' = -1.5$, while the true values of other inputs remain the same. The goal is to update the prediction of the neural network at $x = 1.5$ to this new value, while keeping the predictions of other inputs close to the original predictions, without expensive re-training on the whole dataset. We perform experiments on both EMLP and MLP, showing in Figure 3. Clearly, both neural works successfully update the prediction at $x = 1.5$ to the new value. However, besides the prediction at $x = 1.5$, the learned function of MLP is changed globally while the changes of EMLP are mainly confined in a small local area around $x = 1.5$. That is, we can successfully correct the wrong prediction nearly point-wisely by "editing" the output value for ENNs, but not for classical neural networks. For experiments for class incremental learning and reinforcement learning, please check the appendix.

6. Conclusion

In this work, we proposed elephant activation functions that can make neural networks more resilient to catastrophic forgetting. They are widely applicable and can potentially be combined with other continual learning methods. Specifically, we showed that incorporating elephant activation functions in classical neural networks helps improve continual learning performance. Theoretically, our work provided a deeper understanding of the role of activation functions in catastrophic forgetting. We hope our work will inspire more researchers to study and design better neural network architectures for continual learning.

References

[1] Rahaf Aljundi, Punarjay Chakravarty, and Tinne Tuytelaars. Expert gate: Lifelong learning with a network of experts. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[2] Rahaf Aljundi, Klaas Kelchtermans, and Tinne Tuytelaars. Task-free continual learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019.

[3] Rahaf Aljundi, Marcus Rohrbach, and Tinne Tuytelaars. Selfless sequential learning. In *International Conference on Learning Representations*, 2019.

[4] Davis Blalock, Jose Javier Gonzalez Ortiz, Jonathan Frankle, and John Guttag. What is the state of neural network pruning? *Proceedings of machine learning and systems*, 2020.

[5] Trenton Bricken, Xander Davies, Deepak Singh, Dmitry Krotov, and Gabriel Kreiman. Sparse distributed memory is a continual learner. In *International Conference on Learning Representations*, 2023.

[6] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. OpenAI Gym. *arXiv preprint arXiv:1606.01540*, 2016.

[7] Shuxiao Chen, Hangfeng He, and Weijie Su. Label-aware neural tangent kernel: Toward better generalization and local elasticity. *Advances in Neural Information Processing Systems*, 33: 15847–15858, 2020.

[8] Lenaic Chizat, Edouard Oyallon, and Francis Bach. On lazy training in differentiable programming. *Advances in neural information processing systems*, 2019.

[9] Will Dabney, André Barreto, Mark Rowland, Robert Dadashi, John Quan, Marc G Bellemare, and David Silver. The value-improvement path: Towards better representations for reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2021.

[10] Matthias Delange, Rahaf Aljundi, Marc Masana, Sarah Parisot, Xu Jia, Ales Leonardis, Greg Slabaugh, and Tinne Tuytelaars. A continual learning survey: Defying forgetting in classification tasks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021.

[11] Li Deng. The MNIST database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 2012.

[12] Tim Dettmers and Luke Zettlemoyer. Sparse networks from scratch: Faster training without losing performance. *arXiv preprint arXiv:1907.04840*, 2019.

[13] Mehrdad Farajtabar, Navid Azizan, Alex Mott, and Ang Li. Orthogonal gradient descent for continual learning. In *International Conference on Artificial Intelligence and Statistics*, 2020.

[14] Sebastian Farquhar and Yarin Gal. Towards robust evaluations of continual learning. *arXiv preprint arXiv:1805.09733*, 2018.

[15] Chrisantha Fernando, Dylan Banarse, Charles Blundell, Yori Zwols, David Ha, Andrei A Rusu, Alexander Pritzel, and Daan Wierstra. PathNet: Evolution channels gradient descent in super neural networks. *arXiv preprint arXiv:1701.08734*, 2017.

[16] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *International Conference on Learning Representations*, 2019.

[17] Robert M French. Semi-distributed representations and catastrophic forgetting in connectionist networks. *Connection Science*, 1992.

[18] Robert M French. Catastrophic forgetting in connectionist networks. *Trends in cognitive sciences*, 1999.

[19] Sina Ghiassian, Banafsheh Rafiee, Yat Long Lo, and Adam White. Improving performance in reinforcement learning by breaking generalization in neural networks. In *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems*, 2020.

[20] Yiwen Guo, Anbang Yao, and Yurong Chen. Dynamic network surgery for efficient DNNs. *Advances in neural information processing systems*, 2016.

[21] Hangfeng He and Weijie Su. The local elasticity of neural networks. In *International Conference on Learning Representations*, 2020.

[22] Ching-Yi Hung, Cheng-Hao Tu, Cheng-En Wu, Chien-Hung Chen, Yi-Ming Chan, and Chu-Song Chen. Compacting, picking and growing for unforgetting continual learning. *Advances in Neural Information Processing Systems*, 2019.

[23] Arthur Jacot, Franck Gabriel, and Clément Hongler. Neural tangent kernel: Convergence and generalization in neural networks. *Advances in neural information processing systems*, 2018.

[24] Ronald Kemker, Marc McClure, Angelina Abitino, Tyler Hayes, and Christopher Kanan. Measuring catastrophic forgetting in neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2018.

[25] Khimya Khetarpal, Matthew Riemer, Irina Rish, and Doina Precup. Towards continual reinforcement learning: A review and perspectives. *Journal of Artificial Intelligence Research*, 2022.

[26] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 2017.

[27] Alex Krizhevsky. Learning multiple layers of features from tiny images. *Master's thesis, University of Toronto*, 2009.

[28] Qingfeng Lan, Yangchen Pan, Alona Fyshe, and Martha White. Maxmin Q-learning: Controlling the estimation bias of q-learning. In *International Conference on Learning Representations*, 2020.

[29] Qingfeng Lan, Yangchen Pan, Jun Luo, and A Rupam Mahmood. Memory-efficient reinforcement learning with value-based knowledge consolidation. *Transactions on Machine Learning Research*, 2023.

[30] Lei Le, Raksha Kumaraswamy, and Martha White. Learning sparse representations in reinforcement learning with sparse coding. *International Joint Conferences on Artificial Intelligence*, 2017.

[31] Alexander Li and Deepak Pathak. Functional regularization for reinforcement learning via learned Fourier features. *Advances in Neural Information Processing Systems*, 2021.

[32] Xilai Li, Yingbo Zhou, Tianfu Wu, Richard Socher, and Caiming Xiong. Learn to grow: A continual structure learning framework for overcoming catastrophic forgetting. In *International Conference on Machine Learning*, 2019.

[33] Junjie Liu, Zhe Xu, Runbin Shi, Ray C. C. Cheung, and Hayden K.H. So. Dynamic sparse training: Find efficient sparse network from scratch with trainable masked layers. In *International Conference on Learning Representations*, 2020.

[34] Vincent Liu, Raksha Kumaraswamy, Lei Le, and Martha White. The utility of sparse representations for control in reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2019.

[35] David Lopez-Paz and Marc'Aurelio Ranzato. Gradient episodic memory for continual learning. *Advances in neural information processing systems*, 30, 2017.

[36] Arun Mallya and Svetlana Lazebnik. PackNet: Adding multiple tasks to a single network by iterative pruning. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, 2018.

[37] Marc Masana, Tinne Tuytelaars, and Joost Van de Weijer. Ternary feature masks: zero-forgetting for task-incremental learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021.

[38] Harsh Mehta, Ashok Cutkosky, and Behnam Neyshabur. Extreme memorization via scale of initialization. In *International Conference on Learning Representations*, 2021.

[39] Jorge A Mendez, Harm van Seijen, and Eric Eaton. Modular lifelong reinforcement learning via neural composition. In *International Conference on Learning Representations*, 2022.

[40] Seyed Iman Mirzadeh, Arslan Chaudhry, Dong Yin, Huiyi Hu, Razvan Pascanu, Dilan Gorur, and Mehrdad Farajtabar. Wide neural networks forget less catastrophically. In *International Conference on Machine Learning*, 2022.

[41] Seyed Iman Mirzadeh, Arslan Chaudhry, Dong Yin, Timothy Nguyen, Razvan Pascanu, Dilan Gorur, and Mehrdad Farajtabar. Architecture matters in continual learning. *arXiv preprint arXiv:2202.00275*, 2022.

[42] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, and Ioannis Antonoglou. Playing Atari with deep reinforcement learning. In *NIPS Deep Learning Workshop*, 2013.

[43] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 2015.

[44] Oleksiy Ostapenko, Mihai Puscas, Tassilo Klein, Patrick Jahnichen, and Moin Nabi. Learning to remember: A synaptic plasticity driven framework for continual learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019.

[45] Yangchen Pan, Kirby Banman, and Martha White. Fuzzy tiling activations: A simple approach to learning sparse representations online. In *International Conference on Learning Representations*, 2022.

[46] Matthew Riemer, Ignacio Cases, Robert Ajemian, Miao Liu, Irina Rish, Yuhai Tu, and Gerald Tesauro. Learning to learn without forgetting by maximizing transfer and minimizing interference. In *International Conference on Learning Representations*, 2018.

[47] Andrei A Rusu, Neil C Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks. *arXiv preprint arXiv:1606.04671*, 2016.

[48] Jonathan Schwarz, Wojciech Czarnecki, Jelena Luketina, Agnieszka Grabska-Barwinska, Yee Whye Teh, Razvan Pascanu, and Raia Hadsell. Progress & compress: A scalable framework for continual learning. In *International Conference on Machine Learning*, 2018.

[49] Joan Serra, Didac Suris, Marius Miron, and Alexandros Karatzoglou. Overcoming catastrophic forgetting with hard attention to the task. In *International Conference on Machine Learning*, 2018.

[50] Yang Shen, Sanjoy Dasgupta, and Saket Navlakha. Algorithmic insights on continual learning from fruit flies. *arXiv preprint arXiv:2107.07617*, 2021.

[51] Ghada Sokar, Decebal Constantin Mocanu, and Mykola Pechenizkiy. SpaceNet: Make free space for continual learning. *Neurocomputing*, 2021.

[52] Norman Tasfi. PyGame learning environment. `https://github.com/ntasfi/PyGame-Learning-Environment`, 2016.

[53] Asher Trockman and J Zico Kolter. Patches are all you need? *arXiv preprint arXiv:2201.09792*, 2022.

[54] Gido M Van de Ven and Andreas S Tolias. Three scenarios for continual learning. *arXiv preprint arXiv:1904.07734*, 2019.

[55] Gido M van de Ven, Tinne Tuytelaars, and Andreas S Tolias. Three types of incremental learning. *Nature Machine Intelligence*, 2022.

[56] Liyuan Wang, Xingxing Zhang, Hang Su, and Jun Zhu. A comprehensive survey of continual learning: Theory, method and application. *arXiv preprint arXiv:2302.00487*, 2023.

[57] Cameron R Wolfe and Anastasios Kyrillidis. Cold start streaming learning for deep networks. *arXiv preprint arXiv:2211.04624*, 2022.

[58] Jaehong Yoon, Eunho Yang, Jeongtae Lee, and Sung Ju Hwang. Lifelong learning with dynamically expandable networks. In *International Conference on Learning Representations*, 2018.

[59] Guanxiong Zeng, Yang Chen, Bo Cui, and Shan Yu. Continual learning of context-dependent processing in neural networks. *Nature Machine Intelligence*, 2019.

[60] Friedemann Zenke, Ben Poole, and Surya Ganguli. Continual learning through synaptic intelligence. In *International Conference on Machine Learning*, 2017.

[61] Denny Zhou, Mao Ye, Chen Chen, Tianjian Meng, Mingxing Tan, Xiaodan Song, Quoc Le, Qiang Liu, and Dale Schuurmans. Go wide, then narrow: Efficient training of deep thin networks. In *International Conference on Machine Learning*, 2020.

Appendix A. Function sparsity and gradient sparsity of various activation functions

To begin with, we first define the sparsity of an function, which also applies to activation functions.

Definition 6 (sparse function) *For a function $\sigma : \mathbb{R} \mapsto \mathbb{R}$, we define the sparsity of function σ on input domain $[-C, C]$ as*

$$S_{\epsilon,C}(\sigma) = \frac{|\{x \mid |\sigma(x)| \le \epsilon, x \in [-C, C]\}|}{|\{x \mid x \in [-C, C]\}|} = \frac{|\{x \mid |\sigma(x)| \le \epsilon, x \in [-C, C]\}|}{2C},$$

where ϵ is a small positive number and $C > 0$. As a special case, when $\epsilon \to 0^+$ and $C \to \infty$, define

$$S(\sigma) = \lim_{\epsilon \to 0^+} \lim_{C \to \infty} S_{\epsilon,C}(\sigma).$$

We call σ a $S(\sigma)$-sparse function. In particular, σ is called a sparse function if it is a 1-sparse function, i.e. $S(\sigma) = 1$.

Remark 7 *Easy to verify that $0 \le S(\sigma) \le 1$. The sparsity of a function shows the fraction of nearly zero outputs given a symmetric input domain. For example, both $\mathrm{ReLU}(x)$ and $\mathrm{ReLU}'(x)$ are $\frac{1}{2}$-sparse functions. $\mathrm{Tanh}(x)$ is a 0-sparse function while $\mathrm{Tanh}'(x)$ is a sparse function. In the appendix, we summarize the results for common activation functions in Table 1 and visualize the activation functions with their gradients in Figure 4.*

As shown in Figure 1, elephant activation functions have both sparse function values and sparse gradient values, which can also be formally proved.

Lemma 8 $\mathrm{Elephant}(x)$ *and* $\mathrm{Elephant}'(x)$ *are sparse functions.*

Proof $\mathrm{Elephant}(x) = \frac{1}{1+(\frac{x}{a})^{2d}}$ and $\mathrm{Elephant}'(x) = -\frac{2d}{a}(\frac{x}{a})^{2d-1}(\frac{1}{1+(\frac{x}{a})^{2d}})^2$. For $0 < \epsilon < 1$, easy to verify that

$$|x| \ge a(\frac{1}{\epsilon} - 1)^{1/2d} \Longrightarrow \mathrm{Elephant}(x) \le \epsilon \quad \text{and} \quad |x| \ge \frac{d}{\epsilon} \Longrightarrow \mathrm{Elephant}'(x) \le \epsilon.$$

For $C > a(\frac{1}{\epsilon} - 1)^{1/2d}$, we have $S_{\epsilon,C}(\mathrm{Elephant}) \ge \frac{C - a(\frac{1}{\epsilon}-1)^{1/2d}}{C}$, thus

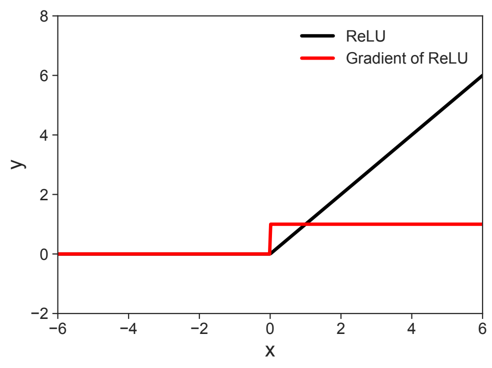
$$S(\mathrm{Elephant}) = \lim_{\epsilon \to 0^+} \lim_{C \to \infty} S_{\epsilon,C}(\mathrm{Elephant}) \ge \lim_{\epsilon \to 0^+} \lim_{C \to \infty} \frac{C - a(\frac{1}{\epsilon} - 1)^{1/2d}}{C} = \lim_{\epsilon \to 0^+} 1 = 1.$$

Similarly, for $C > \frac{d}{\epsilon}$, we have $S_{\epsilon,C}(\mathrm{Elephant}') \ge \frac{C - \frac{d}{\epsilon}}{C}$, thus

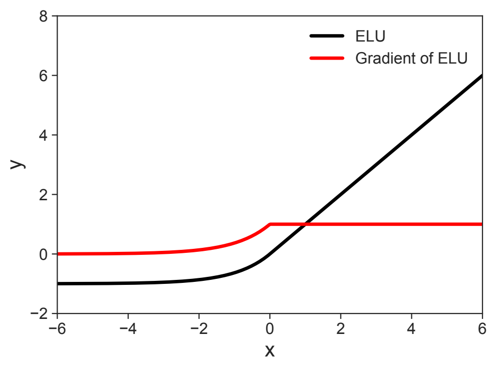
$$S(\mathrm{Elephant}') = \lim_{\epsilon \to 0^+} \lim_{C \to \infty} S_{\epsilon,C}(\mathrm{Elephant}') \ge \lim_{\epsilon \to 0^+} \lim_{C \to \infty} \frac{C - \frac{d}{\epsilon}}{C} = \lim_{\epsilon \to 0^+} 1 = 1.$$

Note that $S(\mathrm{Elephant}) \le 1$ and $S(\mathrm{Elephant}') \le 1$. Together, we conclude that $S(\mathrm{Elephant}) = 1$ and $S(\mathrm{Elephant}') = 1$; $\mathrm{Elephant}(x)$ and $\mathrm{Elephant}'(x)$ are sparse functions. ■

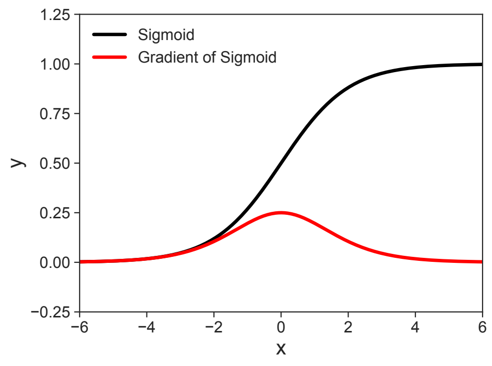
We summarize the function sparsity and gradient sparsity for common activation functions in Table 1 and visualize the activation functions with their gradients in Figure 4.



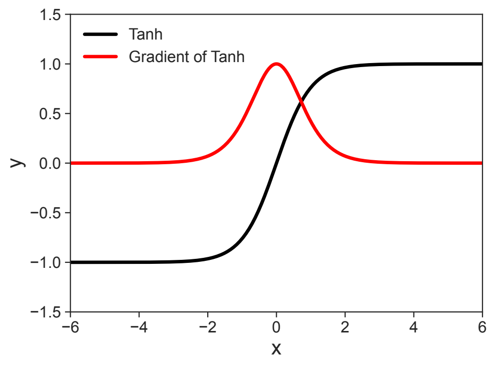
(*a*) ReLU



(*b*) ELU



(*c*) Sigmoid



(*d*) Tanh

Figure 4: Visualizations of common activation functions and their gradients.

Table 1: The function sparsity and gradient sparsity of various activation functions. Among them, only Elephant is sparse in terms of both function values and gradient values.

activation	function sparsity	gradient sparsity
ReLU	1/2	1/2
Sigmoid	1/2	1
Tanh	0	1
ELU	0	1/2
Elephant	1	1

Appendix B. Proofs

Lemma 4 *Given a non-linear function $f_{\boldsymbol{w}}(\boldsymbol{x}) = \boldsymbol{u}^\top \phi_{\boldsymbol{\theta}}(\boldsymbol{x}) : \mathbb{R}^n \mapsto \mathbb{R}$, where $\phi_{\boldsymbol{\theta}}(\boldsymbol{x}) = \sigma(\mathbf{V}\boldsymbol{x} + \boldsymbol{b})$, $\boldsymbol{\theta} = \{\mathbf{V}, \boldsymbol{b}\}$ are learnable parameters, σ is a non-linear activation function, $\boldsymbol{x} \in \mathbb{R}^n$, $\boldsymbol{u} \in \mathbb{R}^m$, $\mathbf{V} \in \mathbb{R}^{m\times n}$, $\boldsymbol{b} \in \mathbb{R}^m$, and $\boldsymbol{w} = \{\boldsymbol{u}, \mathbf{V}, \boldsymbol{b}\}$. The NTK of this non-linear function is*

$$\langle \nabla_{\boldsymbol{w}} f_{\boldsymbol{w}}(\boldsymbol{x}), \nabla_{\boldsymbol{w}} f_{\boldsymbol{w}}(\boldsymbol{x}_t)\rangle = \phi_{\boldsymbol{\theta}}(\boldsymbol{x})^\top \phi_{\boldsymbol{\theta}}(\boldsymbol{x}_t) + \boldsymbol{u}^\top \boldsymbol{u}(\boldsymbol{x}^\top \boldsymbol{x}_t + 1)\phi'_{\boldsymbol{\theta}}(\boldsymbol{x})^\top \phi'_{\boldsymbol{\theta}}(\boldsymbol{x}_t),$$

where $\langle \cdot, \cdot \rangle$ denotes Frobenius inner product or dot product depending on the context.

Proof Let $\circ$ denote Hadamard product. By definition, we have

$$\begin{aligned}
&\langle \nabla_{\boldsymbol{w}} f_{\boldsymbol{w}}(\boldsymbol{x}), \nabla_{\boldsymbol{w}} f_{\boldsymbol{w}}(\boldsymbol{x}_t)\rangle \\
&= \langle \nabla_u f_{\boldsymbol{w}}(\boldsymbol{x}), \nabla_u f_{\boldsymbol{w}}(\boldsymbol{x}_t)\rangle + \langle \nabla_V f_{\boldsymbol{w}}(\boldsymbol{x}), \nabla_V f_{\boldsymbol{w}}(\boldsymbol{x}_t)\rangle + \langle \nabla_b f_{\boldsymbol{w}}(\boldsymbol{x}), \nabla_b f_{\boldsymbol{w}}(\boldsymbol{x}_t)\rangle \\
&= \langle \sigma(\mathbf{V}\boldsymbol{x} + \boldsymbol{b}), \sigma(\mathbf{V}\boldsymbol{x}_t + \boldsymbol{b})\rangle + \langle \boldsymbol{u} \circ \sigma'(\mathbf{V}\boldsymbol{x} + \boldsymbol{b})\boldsymbol{x}^\top, \boldsymbol{u} \circ \sigma'(\mathbf{V}\boldsymbol{x}_t + \boldsymbol{b})\boldsymbol{x}_t^\top\rangle \\
&+ \langle \boldsymbol{u} \circ \sigma'(\mathbf{V}\boldsymbol{x} + \boldsymbol{b}), \boldsymbol{u} \circ \sigma'(\mathbf{V}\boldsymbol{x}_t + \boldsymbol{b})\rangle \\
&= \sigma(\mathbf{V}\boldsymbol{x} + \boldsymbol{b})^\top \sigma(\mathbf{V}\boldsymbol{x}_t + \boldsymbol{b}) + (\boldsymbol{x}^\top \boldsymbol{x}_t + 1)\left(\boldsymbol{u} \circ \sigma'(\mathbf{V}\boldsymbol{x} + \boldsymbol{b})\right)^\top \left(\boldsymbol{u} \circ \sigma'(\mathbf{V}\boldsymbol{x}_t + \boldsymbol{b})\right), \\
&= \sigma(\mathbf{V}\boldsymbol{x} + \boldsymbol{b})^\top \sigma(\mathbf{V}\boldsymbol{x}_t + \boldsymbol{b}) + \boldsymbol{u}^\top \boldsymbol{u}(\boldsymbol{x}^\top \boldsymbol{x}_t + 1)\sigma'(\mathbf{V}\boldsymbol{x} + \boldsymbol{b})^\top \sigma'(\mathbf{V}\boldsymbol{x}_t + \boldsymbol{b}).
\end{aligned}$$

■

Theorem 5 *Define $f_{\boldsymbol{w}}(\boldsymbol{x})$ as it in Theorem 4. Let σ be the elephant activation function with $d \to \infty$. When $|\mathbf{V}(\boldsymbol{x} - \boldsymbol{x}_t)| \succ 2a\mathbf{1}_m$, we have $\langle \nabla_{\boldsymbol{w}} f_{\boldsymbol{w}}(\boldsymbol{x}), \nabla_{\boldsymbol{w}} f_{\boldsymbol{w}}(\boldsymbol{x}_t)\rangle = 0$, where $\succ$ denotes an element-wise inequality symbol and $\mathbf{1}_m = [1, \cdots, 1] \in \mathbb{R}^m$.*

Proof When $d \to \infty$, the elephant function is a rectangular function, i.e.

$$\sigma(x) = \operatorname{rect}(x) = \begin{cases} 1, & |x| < a, \\ \frac{1}{2}, & |x| = a, \\ 0, & |x| > a. \end{cases}$$

In this case, it is easy to verify that $\forall x, y \in \mathbb{R}$, $|x - y| > 2w$, we have $\sigma(x)\sigma(y) = 0$ and $\sigma'(x)\sigma'(y) = 0$. Denote $\Delta_{\boldsymbol{x}} = \boldsymbol{x} - \boldsymbol{x}_t$. Then when $|\mathbf{V}\Delta_{\boldsymbol{x}}| \succ 2a\mathbf{1}_m$, we have $\sigma(\mathbf{V}\boldsymbol{x} + \boldsymbol{b})^\top \sigma(\mathbf{V}\boldsymbol{x}_t + \boldsymbol{b}) = 0$ and $\sigma'(\mathbf{V}\boldsymbol{x} + \boldsymbol{b})^\top \sigma'(\mathbf{V}\boldsymbol{x}_t + \boldsymbol{b}) = 0$. In other words, when $\boldsymbol{x}$ and $\boldsymbol{x}_t$ are dissimilar in the sense that $|\mathbf{V}(\boldsymbol{x} - \boldsymbol{x}_t)| \succ 2a\mathbf{1}_m$, we have $\langle \nabla_{\boldsymbol{w}} f_{\boldsymbol{w}}(\boldsymbol{x}), \nabla_{\boldsymbol{w}} f_{\boldsymbol{w}}(\boldsymbol{x}_t)\rangle = 0$. ■

Appendix C. Related works

Architecture-based continual learning Continual learning methods can be divided into several categories, such as regularization-based methods [3, 26, 48, 60], replay-based methods [10, 14, 24, 54], and optimization-based methods [13, 35, 59]. We encourage readers to check recent surveys [10, 25, 56] for more details. Here, we focus on architecture-based methods that are more related to our work. Among them, our work is inspired by Mirzadeh et al. [40, 41], which study and analyze the effect of different neural architectures on continual learning. Several approaches involve careful selection and allocation of a subset of weights in a large network for each task [15, 32, 36, 37, 49, 51, 58], or the allocation of a network that is specific to each task [1, 47]. Some other methods expand a neural network dynamically [22, 44, 58]. Finally, Shen et al. [50] and Bricken et al. [5] proposed novel neural network structures inspired by biological neural circuits, acting as two strong baseline methods in our experiments.

Sparse representations Sparse representations are known to help reduce forgetting for decades [17]. In supervised learning, sparse training [12, 33, 51] and weight pruning methods [4, 16, 20, 61] are shown to speed up training and improve generalization. In continual learning, both SDMLP [50] and FlyModel [5] are designed to generate sparse representations. In RL, Le et al. [30], Liu et al. [34], and Pan et al. [45] showed that sparse representations help stabilize training and improve overall performance.

Local elasticity and memorization He and Su [21] proposed the concept of local elasticity. Chen et al. [7] introduced label-aware neural tangent kernels, showing that models trained with these kernels are more locally elastic. Mehta et al. [38] proved a theoretical connection between the scale of neural network initialization and local elasticity, demonstrating extreme memorization using large initialization scales. Incorporating Fourier features in the input of a neural network also induces local elasticity, which is greatly affected by the initial variance of the Fourier basis [31].

Appendix D. Experimental details and additional results

For all ENNs, we only replace the activation functions of the last hidden layer with elephant activation functions. [2] Before training, the weight values in all layers are initialized from $U(-\sqrt{k}, \sqrt{k})$, where $k = 1/\text{in\_features}$. For the bias values in the layer where elephant functions are used, we initialize them with evenly spaced numbers over the interval $[-\sqrt{3}\sigma_{bias}, \sqrt{3}\sigma_{bias}]$, where σ_{bias} is the standard deviation of the bias values. The bias values are initialized in such way so that ENNs can generate diverse features. All other bias values are initialized with 0s.

As a limitation, there is a lack of a theoretical way to set σ_{bias} or a in elephant activation functions appropriately. The best value seem to be depend on the input data distribution as well as the number of the input features (i.e., in_features). In practice, we choose σ_{bias} and a from a small set.

To improve sample efficiency, we update a model for multiple times given each new sample/mini-batch data. We call this number the update epoch E.

D.1. Streaming learning for regression

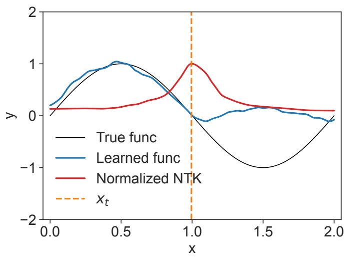
We summarize test MSEs in Table 2. A Lower MSE indicates better performance. For the SR-NN, we present the best result among the combinations of SR-NNs and classical activation functions. Clearly, our method EMLP achieves the best performance, reaching a low test MSE compared with baselines. Generally, we find that the SR-NN achieves slightly better performance than MLPs with classical activation functions, showing the benefits of sparse representations. However, the test MSE of the SR-NN is still large, indicating that the SR-NN fails to approximate $\sin(\pi x)$.

Next, we plot the true function $\sin(\pi x)$, the learned function $f(x)$, and the NTK function $\text{NTK}(x) = \langle \nabla_{\boldsymbol{w}} f_{\boldsymbol{w}}(x), \nabla_{\boldsymbol{w}} f_{\boldsymbol{w}}(x_t) \rangle$ at different training stages for EMLP and SR-NN in Figure 2.

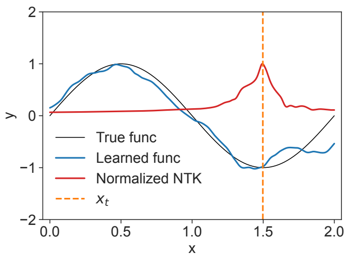
2. Our initial experiments with EMLPs on Split MNIST show that replacing all activation functions with elephant activation functions hurt performance. We leave this for future investigation.

Table 2: The test MSE of various networks in streaming learning for a simple regression task. Lower is better. All results are averaged over 5 runs, reported with standard errors.

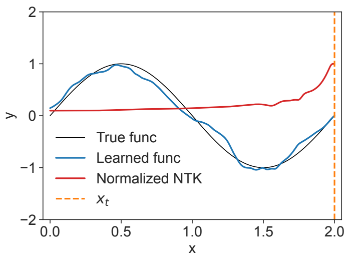
Method	Test Performance (MSE)
MLP (ReLU)	0.4729 ± 0.0110
MLP (Sigmoid)	0.4583 ± 0.0008
MLP (Tanh)	0.4461 ± 0.0013
MLP (ELU)	0.4521 ± 0.0019
SR-NN	0.4061 ± 0.0036
EMLP (Elephant)	$\mathbf{0.0081} \pm \mathbf{0.0009}$



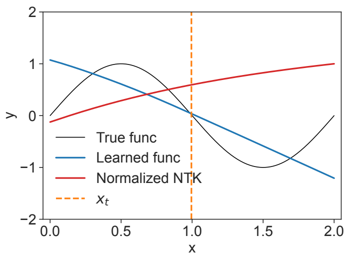
(*a*) EMLP at $t = 100$



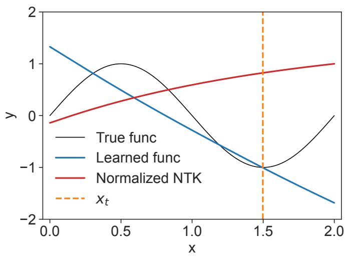
(*b*) EMLP at $t = 150$



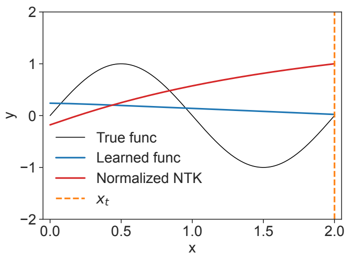
(*c*) EMLP at $t = 200$



(*d*) SR-NN at $t = 100$



(*e*) SR-NN at $t = 150$



(*f*) SR-NN at $t = 200$

Figure 5: Plots of the true function $\sin(\pi x)$, the learned function $f(x)$, and the NTK function $\mathrm{NTK}(x)$ at different training stages for EMLP and SR-NN. The NTK function $\mathrm{NTK}(x)$ of the EMLP quickly reduces to 0 as x moves away from x_t, demonstrating the local elasticity (Property 3) of EMLP.

The plots of MLP (ReLU), MLP (Sigmoid), MLP (Tanh), and MLP (ELU) are not presented, since they are similar to the plots of the SR-NN. The NTK function $\mathrm{NTK}(\boldsymbol{x})$ is normalized such that the function value is in $[-1, 1]$. In Figure 2, the plots in the first row show that for EMLP with $d = 4$, $\mathrm{NTK}(x)$ quickly decreases to 0 as x moving away from x_t, demonstrating the local elasticity (Theorem 3) of EMLP with a small d. However, SR-NNs (and MLPs with classical activation functions) are not locally elastic; the learned function basically evolves as a linear function, a phenomenon often appears in over-parameterized neural networks [8, 23].

We list the (swept) hyper-parameters in streaming learning for regression in Table 3. Among them, d, a, and σ_{bias} are specific hyper-parameters for ENNs, where λ and β are two hyper-parameters used in SR-NN Liu et al. [34].

Table 3: The (swept) hyper-parameters in streaming learning for regression.

Hyper-parameter	Value
Optimizer	Adam
learning rate	$\{3e-3, 1e-3, 3e-4, 1e-4, 3e-5, 1e-5\}$
E	10
d	4
a	$\{0.02, 0.04, 0.08, 0.16, 0.32\}$
σ_{bias}	$\{0.08, 0.16, 0.32, 0.64, 1.28\}$
Set KL loss weight λ	$\{0, 0.1, 0.01, 0.001\}$
SR-NN β	$\{0.05, 0.1, 0.2\}$

D.2. Class incremental learning

In addition to regression tasks, our method can be applied to classification tasks as well, by simply replacing classical activation functions with elephant activation functions in a classification model. Though our model is agnostic to data distributions, we test it in class incremental learning in order to compare it with previous methods. Moreover, we adopt a stricter variation of the continual learning setting by adding the following restrictions: (1) same as streaming learning, each sample only occurs once during training, (2) task boundaries are not provided or inferred [2], (3) neither pre-training nor a fixed feature encoder is allowed [57], and (4) no buffer is allowed to store old task information in any form, such as training samples and gradients.

Surprisingly, we find no methods are designed for or have been tested in the above setting. As a variant of EWC [26], Online EWC [48] almost meets these requirements, although it still requires task boundaries. To overcome this issue, we propose *Streaming EWC* as one of the baselines, which updates the fisher information matrix after every training sample. Streaming EWC can be viewed as a special case of Online EWC, treating each training sample as a new task. Besides Streaming EWC, we consider SDMLP [5] and FlyModel [50] as two strong baselines, although they require either task boundary information or multiple data passes. Finally, two naive baselines are included, which train MLPs and CNNs without any techniques to reduce forgetting.

We test various methods in two standard datasets — Split MNIST [11, 55] and Split CIFAR10 [27, 55]. Both datasets have five sub-datasets, and each of them contains two of the classes. For SDMLP and FlyModel, we take results from Bricken et al. [5] directly. For MLP and CNN, we use ReLU by default. The MLP has only one hidden layer. The CNN model consists of a convolution layer, a max pooling operator, and a linear layer. We also try CNNs with more convolution layers but find the performance is worse and worse as we increase the number of convolution layers. Besides simple CNNs, we test ConvMixer [53] which can achieve 92.5% accuracy in just 25 epochs in classical supervised learning [3]. For our methods, we apply elephant functions with $d = 2$. The resulting neural networks are named EMLP, ECNN, and EConvMixer, respectively.

3. https://github.com/locuslab/convmixer-cifar10

Table 4: The test accuracy of various methods in class incremental learning on Split MNIST. Higher is better. All accuracies are averaged over 5 runs, reported with standard errors. The number of neurons refers to the size of the last hidden layer in a neural network, also known as the feature dimension.

Method	Neurons	Dataset Passes	Task Boundary	Test Accuracy
MLP	1K	1	✗	0.665±0.014
MLP+Streaming EWC	1K	1	✗	0.708±0.008
SDMLP	1K	500	✗	0.69
FlyModel	1K	1	✓	**0.77**
EMLP (ours)	1K	1	✗	0.723±0.006
CNN	1K	1	✗	0.659±0.016
CNN+Streaming EWC	1K	1	✗	0.716±0.024
ECNN (ours)	1K	1	✗	0.732±0.007
ConvMixer	1K	1	✗	0.110±0.003
EConvMixer (ours)	1K	1	✗	0.105±0.003
MLP	10K	1	✗	0.621±0.010
MLP+Streaming EWC	10K	1	✗	0.609±0.013
SDMLP	10K	500	✗	0.53
FlyModel	10K	1	✓	**0.91**
EMLP (ours)	10K	1	✗	0.802±0.002
CNN	10K	1	✗	0.769±0.011
CNN+Streaming EWC	10K	1	✗	0.780±0.010
ECNN (ours)	10K	1	✗	0.850±0.004
ConvMixer	10K	1	✗	0.107±0.003
EConvMixer (ours)	10K	1	✗	0.104±0.003

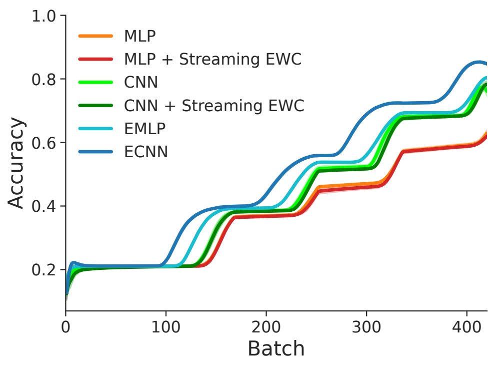
Table 5: The test accuracy of various methods in class incremental learning on Split CIFAR10, averaged over 5 runs. Standard errors are also reported. Higher is better.

Method	Neurons	Test Accuracy
MLP	1K	0.151±0.008
MLP+Streaming EWC	1K	0.158±0.005
EMLP (ours)	1K	**0.197±0.003**
CNN	1K	0.151±0.001
CNN+Streaming EWC	1K	0.147±0.010
ECNN (ours)	1K	**0.192±0.004**
ConvMixer	1K	0.100±0.0027
EConvMixer (ours)	1K	0.100±0.0001
MLP	10K	0.173±0.004
MLP+Streaming EWC	10K	0.169±0.003
EMLP (ours)	10K	**0.239±0.002**
CNN	10K	0.151±0.001
CNN+Streaming EWC	10K	0.179±0.007
ECNN (ours)	10K	**0.243±0.002**
ConvMixer	10K	0.102±0.0015
EConvMixer (ours)	10K	0.100±0.0001

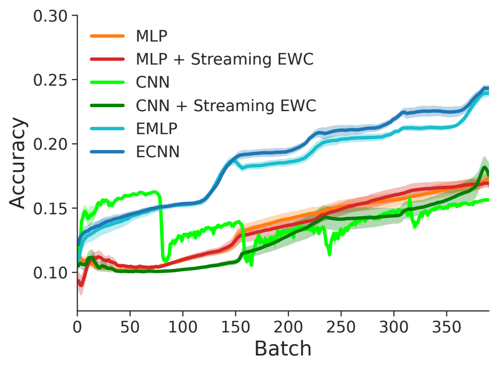
The test accuracy is used as the performance metric. A result summary of different methods on Split MNIST and Split CIFAR10 is shown in Table 4 and Table 5, respectively. In Figure 6, we plot the test accuracy curves during training on Split MNIST and Split CIFAR10 when the number of neurons is $10K$. All results are averaged over 5 runs and the shaded regions represent standard errors. The number of neurons refers to the size of the last hidden layer in a neural network, also known as the feature dimension. Overall, FlyModel performs the best, although it requires additional task boundary information. Our methods (EMLP and ECNN) are the second best without utilizing task boundary information by training for a single pass. Although ConvMixer can achieve great performance in classical supervised learning, it completely fails in class incremental learning setting. Other findings are summarized in the following, reaffirming the findings in Mirzadeh et al. [40, 41]:

- Wider neural networks forget less by using more neurons.
- Neural network architectures can significantly impact learning performance: CNN (ECNN) is better than MLP (EMLP), especially with many neurons.
- Architectural improvement is larger than algorithmic improvement: using a better architecture (e.g., EMLP and ECNN) is more beneficial than incorporating Streaming EWC.
- Using more advanced models does not necessarily lead to better performance in class incremental learning.

Overall, we conclude that applying elephant activation functions significantly reduces forgetting and boosts performance in class incremental learning under strict constraints.



(*a*) Split MNIST



(*b*) Split CIFAR10

Figure 6: The test accuracy curves during training. The x-axis shows the number of training mini-batches. All results are averaged over 5 runs and the shaded regions represent standard errors.

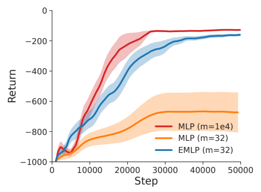
We list the (swept) hyper-parameters for class incremental learning in Table 6. Among them, d, a, and σ_{bias} are specific hyper-parameters for ENNs, where γ and λ are two hyper-parameters used in (streaming) EWC [26].

Table 6: The (swept) hyper-parameters in class incremental learning.

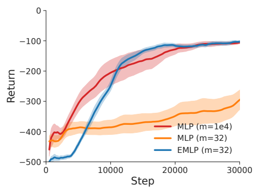
Hyper-parameter	Value
Optimizer	RMSProp with decay=0.999
learning rate	$\{3e-6, 1e-6, 3e-7, 1e-7, 3e-8, 1e-8\}$
mini-batch size	128
E	$\{1, 2\}$
d	2
a	$\{0.02, 0.04, 0.08, 0.16, 0.32\}$
σ_{bias}	$\{0.04, 0.08, 0.16, 0.32, 0.64\}$
EWC γ	$\{0.5, 0.8, 0.9, 0.95, 0.99, 0.999\}$
EWC λ	$\{1e1, 1e2, 1e3, 1e4, 1e5\}$

D.3. Reinforcement Learning

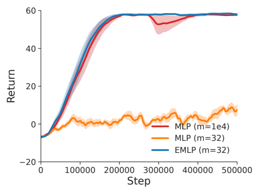
Recently, Lan et al. [29] showed that the forgetting issue exists even in single RL tasks and it is largely masked by a large replay buffer. Without a replay buffer, a single RL task can be viewed as a series of related but different tasks without clear task boundaries [9]. For example, in temporal difference (TD) learning, the true value function is usually approximated by v with bootstrapping: $v(S_t) \leftarrow R_{t+1} + \gamma v(S_{t+1})$, where S_t and S_{t+1} are two successive states and $R_{t+1} + \gamma v(S_{t+1})$ is named the TD target. During training, the TD target constantly changes due to bootstrapping, non-stationary state distribution, and changing policy. To speed up learning while reducing forgetting, it is crucial to update $v(S_t)$ to the new TD target without changing other state values too much [29], where local elasticity can help.



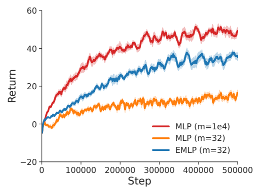
(*a*) MountainCar-v0



(*b*) Acrobot-v1



(*c*) Catcher



(*d*) Pixelcopter

Figure 7: The return curves of DQN in four tasks with different neural networks and buffer sizes. All results are averaged over 10 runs, where the shaded areas represent standard errors. Using a much smaller buffer, EMLP ($m = 32$) outperforms MLP ($m = 32$) and matches MLP ($m = 1e4$).

By its very nature, solving RL tasks requires continual learning ability for both classification and regression. Concretely, estimating value functions using TD learning is a non-stationary regression task; given a specific state, selecting the optimal action from a discrete action space is very similar to a classification task. In this section, we demonstrate that incorporating elephant activation functions helps reduce forgetting in RL under memory constraints. Specifically, we use deep Q-network (DQN) [42, 43] as an exemplar algorithm following Lan et al. [29]. Four classical RL tasks from Gym [6] and PyGame Learning Environment [52] are chosen: MountainCar-v0, Acrobot-v1, Catcher, and Pixelcopter. An MLP/EMLP with one hidden layer of size $1,000$ is used to parameterize the action-value function in DQN for all tasks. For all elephant functions used in EMLPs, $d = 2$. The standard buffer size is 1e4. For EMLP, we use a small replay buffer with size 32; for MLP, we consider two buffer sizes — 1e4 and 32. More details are included in the appendix.

The return curves of various methods are shown in Figure 7, averaged over 10 runs, where shaded areas represent standard errors and m denotes the size of a replay buffer. Clearly, using a tiny replay buffer, EMLP ($m = 32$) outperforms MLP ($m = 32$) in all tasks. Moreover, except Pixelcopter, EMLP ($m = 32$) achieves similar performance compared with MLP ($m = 1e4$), although it uses a much smaller buffer. In summary, the experimental results further confirm the effectiveness of elephant activation functions in reducing catastrophic forgetting.

We list the (swept) hyper-parameters for reinforcement learning in Table 7. Among them, d, a, and σ_{bias} are specific hyper-parameters for ENNs. Other hyper-parameters and training settings can be find in Lan et al. [28].

Table 7: The (swept) hyper-parameters in reinforcement learning.

Hyper-parameter	Value
Optimizer	RMSProp with decay=0.999
learning rate	$\{1e-2, 3e-3, 1e-3, 3e-4, 1e-4, 3e-5, 1e-5, 3e-6\}$
mini-batch size	32
E	$\{1, 2\}$
d	2
a	$\{0.08, 0.16, 0.32, 0.64\}$
σ_{bias}	$\{0.08, 0.16, 0.32, 0.64, 1.28\}$
discount factor γ	0.99