Prefetching Using Principles of Hippocampal-Neocortical Interaction

Michael Wu
Yale University
mw976@yale.edu

Ketaki Joshi
Yale University
ketaki.joshi@yale.edu

Andrew Sheinberg
Yale University
andrew.sheinberg@gmail.com

Guilherme Cox
NVIDIA
gcox@nvidia.com

Anurag Khandelwal
Yale University
anurag.khandelwal@yale.edu

Raghavendra Pradyumna Pothukuchi
Yale University
raghav.pothukuchi@yale.edu

Abhishek Bhattacharjee
Yale University
abhishek@cs.yale.edu

Abstract

Memory prefetching improves performance across many systems layers. However, achieving high prefetch accuracy with low overhead is challenging, as memory hierarchies and application memory access patterns become more complicated. Furthermore, a prefetcher's ability to adapt to new access patterns as they emerge is becoming more crucial than ever. Recent work has demonstrated the use of deep learning techniques to improve prefetching accuracy, albeit with impractical compute and storage overheads. This paper suggests taking inspiration from the learning mechanisms and memory architecture of the human brain—specifically, the hippocampus and neocortex—to build resource-efficient, accurate, and adaptable prefetchers.

CCS Concepts

• **Computer systems organization → Architectures**; **Neural networks**; **Heterogeneous (hybrid) systems**; • **Computing methodologies → Bio-inspired approaches**.

Keywords

Prefetching, Memory Organization, Brain-Inspired Learning

HOTOS '23, June 22–24, 2023, Providence, RI, USA

ACM ISBN 979-8-4007-0195-5/23/06.
https://doi.org/10.1145/3593856.3595901

ACM Reference Format:
Michael Wu, Ketaki Joshi, Andrew Sheinberg, Guilherme Cox, Anurag Khandelwal, Raghavendra Pradyumna Pothukuchi, and Abhishek Bhattacharjee. 2023. Prefetching Using Principles of Hippocampal-Neocortical Interaction. In *Workshop on Hot Topics in Operating Systems (HOTOS '23), June 22–24, 2023, Providence, RI, USA.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3593856.3595901

1 Introduction

Memory prefetching is a performance optimization used widely across several hardware and software layers of modern computer systems. Prefetching proactively brings data from slower levels of memory to faster ones, anticipating its future use. Although well-studied, prefetching continues to be explored, especially as emerging memory hierarchies embrace heterogeneity [22], disaggregation [27], vertical/horizontal tiering [31], and compute in memory [48].

Early prefetchers targeted patterns that were easy to capture, such as strides, and were sufficient for well-understood applications, such as those in SPEC [4]. However, systems and applications today are far more complex and dynamic, rendering simple approaches ineffective. There is a growing interest in developing prefetchers that can adapt to the dynamic execution by *learning* memory access patterns instead of detecting pre-programmed rules [11, 18, 40].

Recent studies have begun exploring the viability of deep learning (DL) for prefetching [11, 18, 30, 40]. In theory, DL should improve prefetching because it is inherently data-driven, and should naturally adapt to applications and their environments. Indeed, these studies show that, in idealized simulations, DL outperforms non-learning prefetch methods in accuracy. However, all of these approaches have three main shortcomings that impede their real-world adoption.

First, the deep neural networks (DNNs) in prior work use impractically high compute and storage resources, even as much as entire applications (§2.1). A more efficient and lightweight learning structure is crucial for real-world systems.

Second, prior approaches train their models offline, without learning continuously from the system execution. This is partly necessary due to the overhead of DNN training. However, such models can then fail to adapt to evolving application phases, configurations, inputs, and other system dynamics. Offline training also requires labeled datasets, which can be impractical to collect at scale.

Third, even if we were to optimize resource usage, online DNNs still face the challenge of catastrophic interference. This issue is well known in machine learning (ML) [23, 28], and describes the tendency of DNNs to drastically forget previously learned information when learning new information.

We consider the fundamental challenge of accurately learning memory access patterns online without interference. We argue that we can develop a robust solution by incorporating principles of learning found in the human brain. We note that the problem of learning access patterns is similar to a task that the human brain encounters continuously, which is to discover generalizable structure from its experiences.

In particular, we take inspiration from the cognitive theory of Complementary Learning Systems (CLS) [32], which models the human brain as an online learning system. CLS theory posits that the brain avoids catastrophic interference using *interleaved replay*, a process where it interleaves the learning of new and old information. Combining this insight with bio-inspired Hebbian networks, which use far less resources than DNNs, helps us build efficient, online prefetchers.

Many principles from CLS have already seen use in the ML community [13, 16, 44, 49]. Their objectives and constraints, however, differ from ours. In addition to maximizing model accuracy, we must further account for metrics like training and inference latency, prefetch timeliness, and usage of memory/network bandwidth. The combination of these constraints motivates a more domain-specific solution.

As we discuss how CLS principles can improve upon DL approaches, we also present expected implementation challenges with two real contexts: disaggregated systems [27] and CPU-GPU systems [3]. We specifically choose these because they experience high cross-node communication latency and operate in relatively resource-constrained settings. Thus, they stand to greatly benefit from intelligent memory prefetching. They also provide the opportunity to implement CLS principles in software, offering a faster path to immediate impact in production systems.

2 DL for Prefetching and Limitations

Prior DL techniques for cache or memory prefetching used transformers [30, 45] or Long Short-Term Memories (LSTMs) [18, 20, 40]. These DNNs use an application's recent cache/memory accesses and instructions to predict its next accesses, and have been shown to achieve high accuracy. However, they have unreasonable resource overheads and are only trained offline, thus remaining only simulated ideals.

Certain works have explored more lightweight learning algorithms for systems, such as reinforcement learning [11], gradient boosting [41], and small DNNs [24]. Unfortunately, these approaches too, were only evaluated in simulation, and we find in our testing that the lightweight models fail to match the prefetching accuracy of larger DNNs. Our goal is to develop online, accurate and resource-efficient prefetchers.

2.1 Overheads of DL-based Prefetching

DL-based prefetching incurs untenable compute and storage overheads. A state-of-the-art LSTM-based cache prefetcher [40] requires over 1 GB of storage using 32-bit parameters. This exceeds the memory capacity of some nodes in our target systems [27, 39]. We model this DNN for memory-page prefetching in a disaggregated system [27], and aggressively compress it to nearly 1 MB by reducing its input-embedding dimension, and the number of output classes. We compile this model on an Intel i7-8700 CPU, and measure its performance. We evaluate CPU performance because, for the inference times we target, which are around 1-10 μs [7, 27, 39], accelerator offloading is not clearly beneficial. Figure 1 shows the prefetcher's deployment.

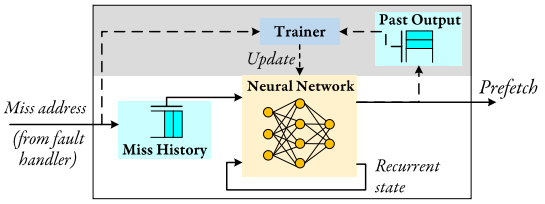
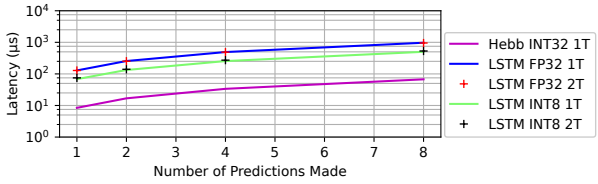
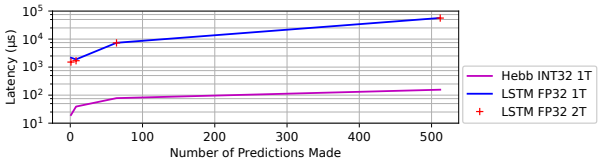


Figure 1: A DNN memory prefetcher. The gray area and dashed lines indicate steps during training.

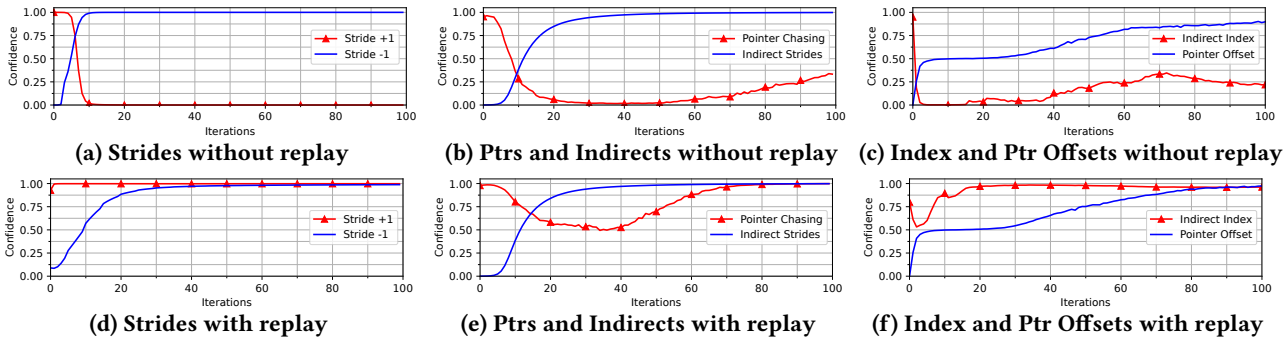


(a) Inference time for various number of future predictions.



(b) Training time for various batch sizes

Figure 2: Inference and training latency of a state-of-the-art LSTM prefetcher on an Intel i7-8700 CPU.



(a) Strides without replay (b) Ptrs and Indirects without replay (c) Index and Ptr Offsets without replay

(d) Strides with replay (e) Ptrs and Indirects with replay (f) Index and Ptr Offsets with replay

Figure 3: Catastrophic interference (a-c) and the effect of replay (d-f) during online prefetch learning. Confidence on the older pattern is shown in red, while the current one is in blue.

Figure 2 shows the LSTM latencies with one and two threads, and with parameter quantization (e.g., FP32 to INT8) during inference [29]. The LSTM takes >150 μs per inference and >1 ms per example for training, which are orders of magnitude higher than our target. Multi-threading is ineffective because LSTMs have poor parallelism [42]. Even after quantization, LSTM inference still takes >60 μs.

Table 1: Memory Access Patterns

Pattern	Code	Behavior
Stride	a[i]	Accessing data at regular delta such as streaming patterns or array traversal.
Pointer chase	*ptr	Pseudorandom accesses to different parts of the memory. E.g. linked list traversal
Indirect stride	*(a[i])	Accessing data at regular delta from a base address. E.g. array of object pointers.
Indirect index	b[a[i]]	Accessing data at indices that are at regular delta from a constant base address.
Pointer offset	*ptr *(ptr+i)	Pseudorandom accesses and adjacent data. E.g. transform on a list/tree.

2.2 Difficulty of Online Prefetch Learning

The scale of modern systems and dynamism of their applications makes it impractical to collect representative datasets with which DNN prefetchers can be trained offline. Instead, it is ideal if the model can adapt to changing memory access behavior by learning online. Unfortunately, learning online makes the DNN vulnerable to *catastrophic interference* [32].

Catastrophic interference occurs when a DNN trained on one pattern begins learning a different, unrelated pattern. The weight updates made when learning the new pattern overwrite the values that were critical in learning the older pattern, causing the DNN to completely forget the older one. Such interference is avoided during offline training by mixing training data and learning over it in multiple passes. This is not the case when learning online.

We use the LSTM from §2.1 to illustrate catastrophic interference with online prefetch learning. We first train the LSTM on a single memory access pattern (e.g., a constant stride) until it achieves perfect accuracy, simulating learning over a single application phase. Next, we present the LSTM with another access pattern (e.g., pointer chasing) to learn. We monitor the LSTM's confidence on the current and previous patterns, which is the probability it assigns to the correct prediction. We select different pairs of patterns from those in Table 1, adapted from prior work [10]. For a pair of patterns, we generate a trace of 1000 accesses with each pattern. We use these *data structure*-level prefetching patterns to avoid confounding effects possible in page-level prefetching.

Figures 3a-3c show our results. As the LSTM learns the current task, its confidence on the older task drops abruptly, demonstrating catastrophic interference. In some cases, the confidence on the first pattern recovers partially, indicating some knowledge transfer. Nonetheless, the final confidence is poor. Such a prefetcher will be ineffective when patterns repeat, which is the case with many applications.

3 Hippocampal-Neocortical Inspired Prefetching

Figure 4 shows the brain's learning architecture in CLS theory. Learning occurs through a complementary relationship between two regions of the brain: the neocortex and the hippocampus. The neocortex, similar to DNNs, slowly learns the structure underlying the information it encounters; i.e., the rules behind a memory access pattern. Meanwhile, the hippocampus quickly memorizes the information it encounters — i.e., the exact memory accesses — in a compressed format, likely by separating each access and storing them in an associative memory [35, 36]. Over time, this information is decompressed and replayed in the neocortex, interleaving the learning of old and new tasks, thus mitigating interference [25, 32]. Furthermore, CLS theory models the networks in the brain using biologically-inspired Hebbian networks, which have much lower resource needs compared to standard DNNs. We show how each of these ideas help overcome the limitations of standard DNNs for prefetching.

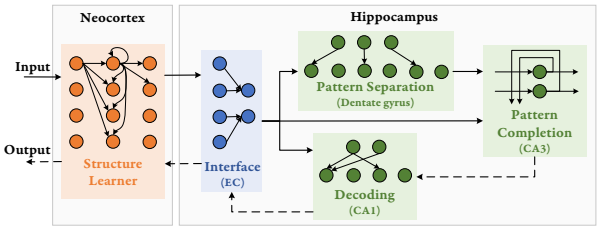


Figure 4: Memory architecture of the brain in CLS theory. Each block is modeled with a Hebbian neural network. Solid lines show information storage paths and dashed lines show recall and replay.

3.1 Overcoming Resource Overheads

Brain-inspired prefetch networks are resource efficient for two reasons. The first reason is that they use a Hebbian update rule, which is much simpler than that of standard DNN learning. When learning with Hebbian networks, a network weight w_{ij}—connecting an input neuron a_i and output neuron b_j—is increased if both neurons are non-zero, and decreased if the input neuron (a_i) is inactive while the output (b_j) is active. The simplified update rule [15, 36] is:

$$\Delta w_{ij} = (b_j \neq 0)[(a_i \neq 0) - (a_i == 0)] \quad (1)$$

This update scheme requires far fewer operations than conventional training of DNNs.

The second advantage of brain-inspired networks comes from their use of sparsity. These networks are sparse in their *connectivity*, meaning a node connects to only 1-25% of the nodes in adjacent layers unlike all-to-all connections in DNNs. Additionally, they are sparse in their *representations*, in that only 1-25% of the network's hidden layer neurons are activated on an input. As a result, the storage and compute needs of these networks are a fraction of those for DNNs.

We prototype a sparse Hebbian neural network for prefetching. The network has a single hidden layer of 1000 neurons, with 12.5% connectivity between layers, and 10% sparsity in the hidden layer. Like an LSTM, our network also uses a recurrent state to capture sequence memory.

Table 2 compares the resource needs of our Hebbian network with that of the LSTM from §2.2. We list the lower bound for the number of operations (Ops) in the LSTM, as its exact value varies with the implementation of transcendental functions (e.g., sigmoid and tanh). The Hebbian network is nearly 3× smaller than the LSTM with nearly an order of magnitude fewer Ops. Hence, the Hebbian training and inference times, shown in Figure 2, are proportionately lower.

Table 2: Resource Needs of Hebbian vs LSTM networks

Model	Parameters	#Ops (inference)	#Ops (training)
LSTM	170 k	> 170k FP	> 400k FP
Hebbian	49 k	14k INT	64k INT

We also compare the networks' prefetching accuracy with multiple applications that have various memory access patterns: TensorFlow [6] training ResNet-50 [19], PageRank [34] using GraphChi [26], mcf [4], and graph500 [1]. For each application, we collect a trace of 2 billion memory accesses and simulate them with a memory sized at 50% of the trace's footprint. We deploy both prefetchers as shown in Figure 1, with a miss history length of 1 input to the networks, and measure the percentage of misses removed compared to a baseline without prefetching. Figure 5 shows the results. Our network has comparable accuracy to the LSTM, even while consuming far fewer resources, showing the effectiveness of Hebbian learning.

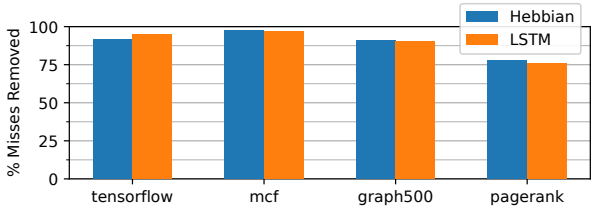


Figure 5: Online memory prefetching performance of Hebbian and LSTM networks.

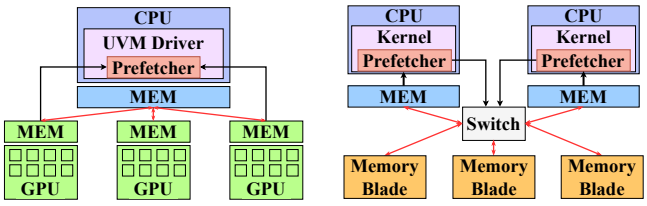
3.2 Overcoming Catastrophic Interference

In CLS theory, catastrophic interference is mitigated by *replaying* past memories stored in the hippocampus. Building a full hippocampal-like storage is an open problem requiring consideration on selectively storing and sampling accesses into it. Therefore, in this work, we will focus on showing the benefits of replay for online prefetch learning without resource limitations on the hippocampal storage.

We perform the experiments in §2.2 that showed catastrophic interference again, but with replay. We implement replay by retraining the network on the first pattern using a 0.1× smaller learning rate after each training/inference of the second. Figures 3d-3f shows the new results. While the models without replay experienced catastrophic interference, performing replay lets the network learn the new pattern *without forgetting* the old one. Even if the old pattern were to repeat, the network would maintain prediction accuracy without needing to re-learn it.

4 Target Systems for Online Prefetching

We target two environments for an initial deployment of our brain-inspired prefetcher: disaggregated systems [27] and CPU-GPU unified virtual memory (UVM) systems [3]. Both systems, shown in Figure 6, have data movement that incurs high communication latencies [7, 27], and stand to benefit from intelligent prefetching. However, the systems differ in a few critical ways that lead to requiring different types of prefetch strategies. One difference is the impact of a page miss. In disaggregated systems, CPU cores fault



(a) UVM system (b) Disaggregated system

Figure 6: Architectures of our target systems. Black lines indicate page miss notifications and prefetch requests. Red lines indicate data movement paths.

only on one page at a time, indicating that the prefetcher should be optimized to hide latency. In GPUs, the SIMT (single instruction multiple thread) execution can produce many concurrent faults, and the lockstep execution model means that a single fault can stall many threads. This suggests that a throughput-optimized prefetcher would be more appropriate for this system.

Our target systems also differ in the location from which the prefetchers can obtain information about memory accesses, and where the compute and memory resources to run prefetching are available. These parameters determine where the prefetcher should be placed. In the UVM system, software-level information on memory accesses is only available in the CPU-located driver. Therefore, all prefetching decisions must ultimately be made from this centralized location. This contrasts with the disaggregated system, where scarce resources on the switch necessitate a decentralized approach with a separate prefetcher per node.

The different placement of the prefetcher in each system, in turn, results in unique design spaces. For example, the prefetcher in the UVM system can take advantage of its global view to make better-informed decisions, but may require more processing to ensure that it can isolate the individual access patterns in the combined access streams. However, such interleaving of access streams may naturally offer more resistance to catastrophic interference, reducing replay costs.

In contrast, the prefetcher in the disaggregated system is less likely to see interleaved access streams since it has a separate prefetcher per CPU. Therefore, the prefetcher network could be smaller to learn the access patterns. With the decentralized architecture, it is easier to integrate application-specific, profile-guided prefetch optimizations.

5 Future Research Challenges

Our initials results with prefetchers based on the hippocampal-neocortical interaction are promising. Harnessing the full benefits of such a prefetcher requires solving important challenges to replicate the CLS architecture in Figure 4 in computer systems. We present these issues, beyond the target system-specific ones discussed in §4.

5.1 Training Instances

Training on every prefetch inference, as done in our experiments (§3.1), can be unnecessary and resource-consuming, especially because training is more expensive than inference. Possible alternatives are to train on a batch of samples at once, and/or to only train on certain misses. Training only on some misses reduces overall training costs, but requires care. Simple approaches, such as randomly deciding which samples to train on, may miss cases that are critical for the model to understand the application. A more intelligent sampling process could use confidence measures from the model to filter less-information carrying samples, or to avoid training on well-learned cases.

5.2 Prefetch Output and Miss History

There are two main parameters for the prefetcher's output: length, which is the number steps predicted into the future; and width, which is the number of predictions made at each step. The ideal values for these parameters are determined by the system architecture and application behavior. Consider our design from §3.1, where the network is trained only to predict the next miss. If the time between misses is less than the inference latency, even a perfect model will always prefetch too late. In that case, a more effective method is to predict a sequence of misses further into the future.

Regarding prefetch width, throughput-bound environments like the UVM system might benefit more from predicting multiple prefetches at a time, even if they have slightly less accuracy. The same could be said for read-heavy workloads. On the other hand, systems where the network is the bottleneck require a prefetcher that is highly selective and confident about bringing in data to minimize communication.

In order to learn how to predict with a given length and width, a prefetcher must maintain a miss history. For example, when prefetching multiple steps into the future, a window of past misses is required to construct appropriate training examples. Thus, the prefetch length determines a minimum history size. Beyond this, the ideal history size depends on the reuse distances in the access patterns of the application. If the current pattern has short reuse distances, then only a few entries in the miss history are necessary, while it is the opposite for patterns with longer reuse distances. Thus, configuring the prefetch length, width, and the access history will require intelligent co-design.

5.3 Encoding Data for Prefetching

Most prior work on ML-based prefetching encodes addresses and strides as one-hot vectors, which are then indexed into an embedding table to obtain a dense representation that is input to the prefetcher [18, 30, 40]. This aligns with approaches used for words in natural language processing, but storing

embeddings can become expensive (e.g., >500 MB [40]). It also inflates compute costs, as the output layer grows linearly with the number of embedding vectors.

A more fundamental issue is that addresses and strides can be a poor proxy for understanding the inherent behavior of an application. We found that, as of now, neither the LSTM nor the Hebbian network perform well on caching applications like memcached [17] and cachebench [12]. This is because these applications are almost entirely pointer-based, and the access patterns are difficult to learn from addresses or strides.

Ideally, the representation of the input data should more closely resemble how addresses "flow" at the data structure level (e.g. tree nodes, pointer chains). We have found that insights from the cognitive theories can provide inspiration here as well. There is evidence that the hippocampus encodes locations optimizing for vector-based navigation [33]. A similar encoding for addresses could better represent paths through data structures. Other brain-inspired work has explored ways of representing symbols that allow the efficient detection of relations in neural networks [8]. An analog of such an approach for prefetching would be an address embedding optimized for detecting pointers that are logically (as opposed to numerically) close to one another.

5.4 Implementing Replay

In our experiments to demonstrate the utility of replay, we assumed that we could store all past examples, and interleave them later. A full implementation must trade the storage/-compute costs of replay with its benefits for learning. One simple approach is to use a fixed-size buffer. This, however, could lose important information as entries are evicted. A more principled approach could save space by filtering less important examples, perhaps using confidence as a measure, or freeing entries that have already been consolidated due to replay, thus not needed further learning [32]. Yet another alternative is to average similar examples, producing single representative cases.

Another challenge in incorporating replay is to define application phases so that they can be replayed. However, phase characteristics can vary significantly between applications, making it difficult to manage replay with a single parameter setting (buffer size, time limits, miss counts, etc.). This could motivate an interface for application developers to directly tune replay parameters, or to indirectly indicate phase behavior and timings. Another approach, also inspired by cognitive theories, is to identify contexts or phases using clustering of abstract representations learned by the network [14].

Finally, our preliminary studies also show only one form of replay. Cognitive literature describes many forms of replay [46], each with their own benefits and challenges. Some methods such as *hindsight* or *simulation* replay, where the prefetcher artificially generates memory sequences and learns them, can be helpful in avoiding replay storage costs altogether. Such sequences could be generated by rules that are determined with profile-guided techniques, or through generative networks i.e., trading off compute for memory. Another alternative could generate hidden layer values, complete the forward pass, and train on the output to reinforce existing behavior. We leave such ideas for future research.

5.5 Availability

Since training actively changes the weights of a neural network, it may be important to block inference during training. This is an availability issue, which motivates a protocol where training is applied to a separate model copy, which is later redeployed when the live model's confidence/accuracy decreases. However, it is possible that simpler approaches could also suffice for two reasons. One, because prefetching is not correctness-critical, inference requests are safe to drop. Second, neural networks can have noise-robust or noise-smoothing effects, meaning that small perturbations to weights do not cause large changes in the network's output, especially since our training method explicitly seeks to preserve the network's prior performance [21, 35]. This could allow inference to remain accurate even when concurrent with training. We expect this property to require further study, as it could significantly optimize a real prefetcher deployment, but may be sensitive to the many details of practical implementation.

6 Conclusion

Recent deep learning approaches to prefetching have shown promising results, but only in idealized simulations. Their real implementation is impeded due to resource overheads and learning limitations. This paper explores how one might address these issues using principles and models of the hippocampus and neocortex, as well as some challenges we should expect in implementing them. We ultimately expect more challenges to appear in designing and deploying such models, but we hope this paper serves as a starting point for implementing efficient and intelligent learning algorithms for all relevant systems contexts.

7 Acknowledgements

We thank Jonathan D. Cohen for his insights on complementary learning systems. We also thank Seung-seob Lee for his help with generating the traces used in this work. This work was made possible in part via funding from the National Science Foundation awards 2118851, 2040682, 2112562, 2047220 and a Computing Innovation Fellowship for Raghavendra Pradyumna Pothukuchi (under NSF grant 2127309 to the Computing Research Association).

References

[1] 2010. Graph 500 | large-scale benchmarks. https://graph500.org/.

[2] 2016. Introduction to Cache Allocation Technology in the Intel® Xeon® Processor E5 v4 Family. https://www.intel.com/content/www/us/en/developer/articles/technical/introduction-to-cache-allocation-technology.html.

[3] 2017. Nvidia Unified Memory. https://developer.nvidia.com/blog/unified-memory-cuda-beginners/.

[4] 2017. SPEC Benchmarks and Tools. https://www.spec.org/benchmarks.html.

[5] 2018. libtorch. https://github.com/pytorch/pytorch/blob/master/docs/libtorch.rst.

[6] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. 2016. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467* (2016).

[7] Tyler Allen and Rong Ge. 2021. Demystifying GPU UVM Cost with Deep Runtime and Workload Analysis. In *2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. 141–150. https://doi.org/10.1109/IPDPS49936.2021.00023

[8] Awni Altabaa, Taylor Webb, Jonathan Cohen, and John Lafferty. 2023. Abstractors: Transformer Modules for Symbolic Message Passing and Relational Reasoning. *arXiv preprint arXiv:2304.00195* (2023).

[9] Berk Atikoglu, Yuehai Xu, Eitan Frachtenberg, Song Jiang, and Mike Paleczny. 2012. Workload Analysis of a Large-Scale Key-Value Store. In *Proceedings of the 12th ACM SIGMETRICS/PERFORMANCE Joint International Conference on Measurement and Modeling of Computer Systems* (London, England, UK) *(SIGMETRICS '12)*. Association for Computing Machinery, New York, NY, USA, 53–64. https://doi.org/10.1145/2254756.2254766

[10] Grant Ayers, Heiner Litz, Christos Kozyrakis, and Parthasarathy Ranganathan. 2020. Classifying Memory Access Patterns for Prefetching. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems* (Lausanne, Switzerland) *(ASPLOS '20)*. Association for Computing Machinery, New York, NY, USA, 513–526. https://doi.org/10.1145/3373376.3378498

[11] Rahul Bera, Konstantinos Kanellopoulos, Anant Nori, Taha Shahroodi, Sreenivas Subramoney, and Onur Mutlu. 2021. Pythia: A Customizable Hardware Prefetching Framework Using Online Reinforcement Learning. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture* (Virtual Event, Greece) *(MICRO '21)*. Association for Computing Machinery, New York, NY, USA, 1121–1137. https://doi.org/10.1145/3466752.3480114

[12] Benjamin Berg, Daniel S. Berger, Sara McAllister, Isaac Grosof, Sathya Gunasekar, Jimmy Lu, Michael Uhlar, Jim Carrig, Nathan Beckmann, Mor Harchol-Balter, and Gregory R. Ganger. 2020. The CacheLib Caching Engine: Design and Experiences at Scale. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. USENIX Association, 753–768. https://www.usenix.org/conference/osdi20/presentation/berg

[13] Prabuddha Chakraborty and Swarup Bhunia. 2022. BINGO: brain-inspired learning memory. *Neural Computing and Applications* 34 (02 2022), 1–25. https://doi.org/10.1007/s00521-021-06484-8

[14] Anne GE Collins and Michael J Frank. 2013. Cognitive control over learning: creating, clustering, and generalizing task-set structure. *Psychological review* 120, 1 (2013), 190.

[15] Georgios Detorakis, Travis Bartley, and Emre Neftci. 2019. Contrastive Hebbian learning with random feedback weights. *Neural Networks* 114 (2019), 1–14. https://doi.org/10.1016/j.neunet.2019.01.008

[16] William Fedus, Prajit Ramachandran, Rishabh Agarwal, Yoshua Bengio, Hugo Larochelle, Mark Rowland, and Will Dabney. 2020. Revisiting Fundamentals of Experience Replay. In *Proceedings of the 37th International Conference on Machine Learning (ICML'20)*. JMLR.org, Article 287, 11 pages.

[17] Brad Fitzpatrick. 2004. Distributed caching with memcached. *Linux Journal* (August 2004).

[18] Milad Hashemi, Kevin Swersky, Jamie Smith, Grant Ayers, Heiner Litz, Jichuan Chang, Christos Kozyrakis, and Parthasarathy Ranganathan. 2018. Learning Memory Access Patterns. In *Proceedings of the 35th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 80)*, Jennifer Dy and Andreas Krause (Eds.). PMLR, 1919–1928.

[19] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.

[20] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-Term Memory. *Neural Comput.* 9, 8 (nov 1997), 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735

[21] Heiko Hoffmann. 2019. Sparse Associative Memory. *Neural Computation* 31, 5 (05 2019), 998–1014. https://doi.org/10.1162/neco_a_01181 arXiv:https://direct.mit.edu/neco/article-pdf/31/5/998/1052590/neco_a_01181.pdf

[22] Sudarsun Kannan, Ada Gavrilovska, Vishal Gupta, and Karsten Schwan. 2017. HeteroOS: OS Design for Heterogeneous Memory Management in Datacenter. *SIGARCH Comput. Archit. News* 45, 2 (jun 2017), 521–534. https://doi.org/10.1145/3140659.3080245

[23] Ronald Kemker, Marc McClure, Angelina Abitino, Tyler Hayes, and Christopher Kanan. 2018. Measuring catastrophic forgetting in neural networks. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 32.

[24] Tim Kraska, Alex Beutel, Ed H. Chi, Jeffrey Dean, and Neoklis Polyzotis. 2018. The Case for Learned Index Structures. In *Proceedings of the 2018 International Conference on Management of Data* (Houston, TX, USA) *(SIGMOD '18)*. Association for Computing Machinery, New York, NY, USA, 489–504. https://doi.org/10.1145/3183713.3196909

[25] Dharshan Kumaran, Demis Hassabis, and James L McClelland. 2016. What Learning Systems do Intelligent Agents Need? Complementary Learning Systems Theory Updated. *Trends in Cognitive Sciences* 20, 7 (2016), 512–524. https://doi.org/10.1016/j.tics.2016.05.004

[26] Aapo Kyrola, Guy Blelloch, and Carlos Guestrin. 2012. GraphChi: Large-Scale Graph Computation on Just a PC. In *Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation* (Hollywood, CA, USA) *(OSDI'12)*. USENIX Association, USA, 31–46.

[27] Seung-seob Lee, Yanpeng Yu, Yupeng Tang, Anurag Khandelwal, Lin Zhong, and Abhishek Bhattacharjee. 2021. MIND: In-Network Memory Management for Disaggregated Data Centers. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles* (Virtual Event, Germany) *(SOSP '21)*. Association for Computing Machinery, New York, NY, USA, 488–504. https://doi.org/10.1145/3477132.3483561

[28] Sang-Woo Lee, Jin-Hwa Kim, Jaehyun Jun, Jung-Woo Ha, and Byoung-Tak Zhang. 2017. Overcoming catastrophic forgetting by incremental moment matching. *Advances in neural information processing systems* 30 (2017).

[29] Hao Li, Soham De, Zheng Xu, Christoph Studer, Hanan Samet, and Tom Goldstein. 2017. Training Quantized Nets: A Deeper Understanding. In *Advances in Neural Information Processing Systems*, I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.), Vol. 30. Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2017/file/1c303b0eed3133200cf715285011b4e4-Paper.pdf

[30] Xinjian Long, Xiangyang Gong, Bo Zhang, and Huiyang Zhou. 2023. Deep learning based data prefetching in CPU-GPU unified virtual memory. *J. Parallel and Distrib. Comput.* 174 (Apr 2023), 19–31. https://doi.org/10.1016/j.jpdc.2022.12.004

[31] Hasan Al Maruf, Hao Wang, Abhishek Dhanotia, Johannes Weiner, Niket Agarwal, Pallab Bhattacharya, Chris Petersen, Mosharaf Chowdhury, Shobhit Kanaujia, and Prakash Chauhan. 2022. Tpp: Transparent page placement for cxl-enabled tiered memory. *arXiv preprint arXiv:2206.02878* (2022).

[32] James L McClelland, Bruce L McNaughton, and Randall C O'Reilly. 1995. Why there are complementary learning systems in the hippocampus and neocortex: insights from the successes and failures of connectionist models of learning and memory. *Psychological review* 102, 3 (1995), 419.

[33] B. L. McNAUGHTON, C. A. Barnes, J. L. Gerrard, K. Gothard, M. W. Jung, J. J. Knierim, H. Kudrimoti, Y. Qin, W. E. Skaggs, M. Suster, and K. L. Weaver. 1996. Deciphering The Hippocampal Polyglot: the Hippocampus as a Path Integration System. *Journal of Experimental Biology* 199, 1 (01 1996), 173–185. https://doi.org/10.1242/jeb.199.1.173 arXiv:https://journals.biologists.com/jeb/article-pdf/199/1/173/2532105/jexbio_199_1_173.pdf

[34] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. 1999. *The PageRank citation ranking: Bringing order to the web.* Technical Report. Stanford InfoLab.

[35] Edmund Rolls. 2018. The storage and recall of memories in the hippocampo-cortical system. *Cell and Tissue Research* 373 (09 2018). https://doi.org/10.1007/s00441-017-2744-3

[36] Edmund T. Rolls. 2016. Pattern separation, completion, and categorisation in the hippocampus and neocortex. *Neurobiology of Learning and Memory* 129 (2016), 4–28. https://doi.org/10.1016/j.nlm.2015.07.008 Pattern Separation and Pattern Completion in the Hippocampal System.

[37] Geet Sethi, Bilge Acun, Niket Agarwal, Christos Kozyrakis, Caroline Trippel, and Carole-Jean Wu. 2022. RecShard: Statistical Feature-Based Memory Optimization for Industry-Scale Neural Recommendation. In *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems* (Lausanne, Switzerland) *(ASPLOS '22)*. Association for Computing Machinery, New York, NY, USA, 344–358. https://doi.org/10.1145/3503222.3507777

[38] Mohammad Shahrad, Rodrigo Fonseca, Íñigo Goiri, Gohar Chaudhry, Paul Batum, Jason Cooke, Eduardo Laureano, Colby Tresness, Mark Russinovich, and Ricardo Bianchini. 2020. Serverless in the Wild: Characterizing and Optimizing the Serverless Workload at a Large Cloud Provider. In *Proceedings of the 2020 USENIX Conference on Usenix Annual Technical Conference (USENIX ATC'20)*. USENIX Association, USA, Article 14, 14 pages.

[39] Yizhou Shan, Yutong Huang, Yilun Chen, and Yiying Zhang. 2018. Legoos: A disseminated, distributed {OS} for hardware resource disaggregation. In *13th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 18)*. 69–87.

[40] Zhan Shi, Akanksha Jain, Kevin Swersky, Milad Hashemi, Parthasarathy Ranganathan, and Calvin Lin. 2021. A Hierarchical Neural Model of Data Prefetching. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems* (Virtual, USA) *(ASPLOS '21)*. Association for Computing Machinery, New York, NY, USA, 861–873. https://doi.org/10.1145/3445814.3446752

[41] Zhenyu Song, Daniel S. Berger, Kai Li, and Wyatt Lloyd. 2020. Learning Relaxed Belady for Content Distribution Network Caching. In *Proceedings of the 17th Usenix Conference on Networked Systems Design and Implementation* (Santa Clara, CA, USA) *(NSDI'20)*. USENIX Association, USA, 529–544.

[42] Marijn F. Stollenga, Wonmin Byeon, Marcus Liwicki, and Juergen Schmidhuber. 2015. Parallel Multi-Dimensional LSTM, with Application to Fast Biomedical Volumetric Image Segmentation. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2* (Montreal, Canada) *(NIPS'15)*. MIT Press, Cambridge, MA, USA, 2998–3006.

[43] Elvira Teran, Zhe Wang, and Daniel A. Jiménez. 2016. Perceptron learning for reuse prediction. In *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 1–12. https://doi.org/10.1109/MICRO.2016.7783705

[44] Simon Thomann, Paul R. Genssler, and Hussam Amrouch. 2022. HW/SW Co-design for Reliable In-memory Brain-inspired Hyperdimensional Computing. https://doi.org/10.48550/ARXIV.2202.04789

[45] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems* 30 (2017).

[46] Lennart Wittkuhn, Samson Chien, Sam Hall-McMaster, and Nicolas W. Schuck. 2021. Replay in minds and machines. *Neuroscience & Biobehavioral Reviews* 129 (2021), 367–388. https://doi.org/10.1016/j.neubiorev.2021.08.002

[47] Juncheng Yang, Yao Yue, and K. V. Rashmi. 2021. A Large-Scale Analysis of Hundreds of In-Memory Key-Value Cache Clusters at Twitter. *ACM Trans. Storage* 17, 3, Article 17 (aug 2021), 35 pages. https://doi.org/10.1145/3468521

[48] Shimeng Yu, Wonbo Shim, Xiaochen Peng, and Yandong Luo. 2021. RRAM for compute-in-memory: From inference to training. *IEEE Transactions on Circuits and Systems I: Regular Papers* 68, 7 (2021), 2753–2765.

[49] Xuejiao Zhao, Huanhuan Chen, Zhenchang Xing, and Chunyan Miao. 2020. Brain-inspired Search Engine Assistant based on Knowledge Graph. https://doi.org/10.48550/ARXIV.2012.13529