Understanding Memory Access Patterns for Prefetching

Peter Braun
Computer Science and Engineering
University of California, Santa Cruz
Santa Cruz, California
pvbraun@ucsc.edu

Heiner Litz
Computer Science and Engineering
University of California, Santa Cruz
Santa Cruz, California
hlitz@ucsc.edu

***Abstract*—The Von Neumann bottleneck is a persistent problem in computer architecture, causing stalls and wasted CPU cycles. The Van Neumann bottleneck is particularly relevant for memory-intensive workloads whose working set does not fit into the microprocessor's cache and hence memory accesses suffer the high access latency of DRAM. One technique to address this bottleneck is to prefetch data from memory into on-chip caches. While prefetching has proven successful, for simple access patterns such as strides, existing prefetchers are incapable of providing benefit for applications with complex, irregular access patterns. A neural network-based prefetcher shows promise for these challenging workloads.**

We provide a better understanding of what type of memory access patterns an LSTM neural network can learn by training individual models on microbenchmarks with well-characterized memory access patterns. We explore a range of model parameters and provide a better understanding of what model is ideal to use. We achieve over 95% accuracy on the microbenchmarks and find a strong relationship between lookback (history window) size and the ability of the model to learn the pattern. We find also an upper limit on the number of concurrent distinct memory access streams that can be learned by a model of a given size.

I. Introduction

The Von Neumann performance bottleneck is a well-known and persistent problem within computer architecture. The latency of an access to DRAM can cause the processor to stall for many cycles, a significant inefficiency. Many techniques have been implemented to address this problem, with the most important being the use of small, fast caches close to the processor. The caches exploit the spatial and temporal memory access locality exhibited by most programs to reduce the latency of an access. A data prefetcher can improve the utility of caches, by predicting what data will be used in the near future, fetching it from DRAM into the cache. Existing mechanisms such as the stride or GHB prefetcher are unable to perform well to prefetch complex memory access patterns such as those that are irregular. This includes memory-intensive applications such as graph processing, applications that spend a significant amount traversing pointer based data structures as well as datacenter applications that exhibit large working sets exceeding the processor caches [1].

Previous work [3] has found promise for the use of a long short term memory (LSTM) deep neural network (DNN) for prediction of memory accesses in complex memory-intensive workloads. While the work has shown good prediction accuracy on the Spec2006 benchmark suite [5], it fails to provide an in-depth analysis of how well DNNs can predict different types of access patterns. To address this knowledge gap we develop the following methodology. We first determine a set of microbenchmarks which cover common memory access patterns. Next, we execute the microbenchmarks, tracing their memory access patterns. We then train LSTM based DNN models for each of the microbenchmark traces, to gain an in-depths understanding of the types of memory accesses that can be predicted with good performance. As part of this work, we make the observation that a key technique for achieving high accuracy is proper data preparation. Finally, we evaluate a range of DNN hyperparameters to determine the effect of model complexity on prediction accuracy. We find that our DNN model achieves high accuracy for next-element prediction in all of our microbenchmarks, given generous enough hyperparameters. We find also that appropriate selection of window size of previous loads (lookback) plays a key role in allowing the model to capture and identify local patterns, and that the size of the DNN model can be substantially decreased for moderately complex access patterns.

This work is supported by Intel and the NSF grant: CCF-1823559

II. Background

A. Microarchitectural Prefetchers

Data prefetchers are used to prefetch data that is expected to be used soon from DRAM into caches. When the prediction is correct then the latency of that memory access can be dramatically decreased. A typical access time can be 200 CPU cycles for DRAM and 40 cycles for the L3 cache, providing a potential latency reduction of 5x. When prefetching into the L1 cache directly this can increase to up to 100x. Most popular prefetchers such as GHB [9] and stride use recent memory accesses to predict future accesses. These prefetchers are hardware-based and generally follow simple heuristics or algorithms. A typical stride prefetcher can identify a fixed number, e.g. 16 separate streams. In the case workloads concurrently utilize a greater number of streams, prefetcher performance starts to decline as the many streams compete for resources such as memory access history tables. This defines a limit on the complexity of the memory access patterns

that can be handled. More sophisticated prefetchers have the potential to be able to capture more complex patterns, however, generally only work well for a subset of applications. The main issue with these prefetchers is that they generally apply a single workload-independent technique that needs to work well in average. Machine learning approaches that can train application specific models show promise in addressing this limitation.

B. Deep Neural Networks

Recent advances in deep neural networks have driven their success in fields such as natural language processing, artificial language processing and image recognition. DNNs are now being explored for a wide range of problems. Their strength lies in the backpropagation algorithm that enables convergence to local optima efficiently. DNNs obtain their prediction by taking a numeric input feature vector and computing across layers of "neurons" to provide an output vector which is interpreted to provide the prediction. The approach can be smaller in terms of space and computation compared to manually designed rule-based models [2].

The NLP problem of predicting the next word in a sentence seems like a particularly interesting problem related to the prefetching challenge. The task is, given a sequence of N most recent words, predict the subsequent word. In NLP sequence prediction, there also exist complex patterns and interrelationships between words in the sentence, similarly as there exist relations between the memory accesses emitted by an application.

A successful type of DNN used for this problem is the LSTM long short-term memory network [8]. The LSTM is a type of recurrent neural net, designed for time-series sequences, and provides the benefits of smaller model size due to weight sharing. LSTMs contain memory elements storing information of the past, controlled by a forget gate whose weights are also learned as part of the training process. This has the potential for better capturing longer term relationships between inputs. A DNN generally consists of multiple layers, whereas the number of layers is defined as the depth of the network. Each layer has a certain width, defined by its tensor that can be tuned by the model developer. Finally, LSTMs define a lookback parameter which, in the scope of this work, defines the number of past accesses the model considers to predict the next.

III. Problem Formulation

The prefetching problem can be formulated as a prediction problem. Given the past N memory accesses, predict the next access. Common features used by current prefetchers are the sequence of the N recent memory addresses as well as their associated instructions defined by the program counter (PCs) address. Each PC is uniquely associated with a particular instruction. A given load instruction will often have a more predictable sequence of memory accesses, so knowing the PC can help distinguish separate patterns or streams.

More specifically, data prefetching can be seen as a classification problem. From a pool of the k most common memory addresses, choose the most likely address to occur next. The drawback of this approach is that the number of memory addresses accessed can be very large, prompting the question of whether there is a way of decreasing the size of the state space. A second option is be to use the memory address deltas, defined as the difference between address N and address $N + 1$. Typical memory access patterns including array or immutable list traversals, contain far fewer deltas than distinct memory addresses. Contemporary stride and GHB prefetchers exploit this characteristic for the same reason, increasing prediction accuracy.

IV. Microbenchmark Motivation

Little is understood about how well DNNs predict certain access patterns and finding performance on a full program obscures what the DNN is able to learn. To better understand this we created a suite of microbenchmarks each one designed to exhibit a particular memory access pattern that mimics small parts of a real world program. The key patterns chosen are 'strided', 'periodic', and linked-list traversals. A strided pattern has a single recurring delta between successive memory addresses. An example of this is an array traversal. A periodic pattern has a repeating period, a sequence of deltas that repeats. This type of access pattern can be created when accessing fields within elements of a data structure. The last pattern is the linked-list traversal. When the linked-list is not modified often this pattern can be thought of a sub-type of the periodic pattern: during each traversal nodes are accessed beginning from the first and deltas between each successive node may be arbitrary. The ubiquity of linked-lists within many applications make this an essential target for prefetching prediction.

To better understand the limitations of the DNN model for memory access prediction, it is evaluated with these fundamental patterns and progressively more complex compositions of these patterns.

V. Methodology

In this section we describe our methodology of developing microbenchmarks, generating memory traces as well as cache miss profiles. Furthermore, we show how traces are preprocessed and how we train our predictive DNN models.

A. Microbenchmarks

We generated a suite of microbenchmarks written in C++, representing the different memory access patterns discussed in the previous section. In particular, we developed the following applications:

- Array traversal (sequential memory accesses)
- Array of structs (strided memory access with configurable distance)
- Traversal of a fixed length immutable list (periodic accesses)
- Compositions of multiple access patterns

The strided patterns continually accessed a memory location that was n bytes away. A multiple of 64 bytes was chosen to prevent successive accesses to the same cache line and create a more interesting pattern to predict. The periodic patterns each had a fixed sequence of 5-7 deltas that was repeated, each again multiples of 64 bytes.

The first 3 patterns are 2 simple patterns that are interleaved: 2 strided, 1 strided and 1 periodic, and 2 periodic. The program switches to the next pattern after 40 data accesses. The next are variations of 4 interleaved periodic patterns that switch patterns every 20 accesses. In the first, the switch is regular, e.g. pattern1, pattern2, pattern3, pattern4, pattern1, ... In the next, the next access pattern is chosen randomly. For these with the random switch, a variable amount of labelled noise was added, a second PC that performed random memory accesses. The next microbenchmarks increase the complexity of the trace by scaling up the number of interleaved periodic patterns to up to 1000 separate streams. The last microbenchmark consists of pointer-chasing, linked list traversals where elements are looked up. This access pattern would manifest similarly to the periodic pattern: the deltas between each node start address could be random, but the pattern is repeated continuously.

B. Trace Generation

To obtain the input for model training, we execute the microbenchmarks and obtain memory access traces via Dynamorio's memtrace [6]. Memtrace captures the instruction program counters (PC) of all executed basic blocks (BBLs) as well as the effective addresses of all loads and stores. In combination with the binary executable, traces can accurately replay all instructions and memory accesses executed by the program. We then feed the obtained traces to the zsim [7] microarchitectural simulator to perform cache simulation. In particular, we capture all L1 cache accesses as well as all L3 misses simultaneously. Tracking L3 misses provides information about the memory accesses which are most valuable to prefetch, while capturing L1 accesses carries precise information about memory accesses which would be lost if we were only focusing on L3 misses. The output of this step is a sequence of 3-tuples containing the memory address, PC of the load/store instruction as well as L3 miss information. This obtained sequence is further pre-processed before feeding it to DNN training.

C. Data Preprocessing

As shown by [3] predicting absolute memory addresses is difficult due to the size and sparsity of the 64-bit memory address space. A promising technique, therefore, is to compute memory address deltas of the absolute addresses. As the number of deltas, e.g. for a stride access pattern, is much lower than the number of absolute addresses, prediction accuracy can be improved. Prior work computed deltas between consecutive memory accesses [3] [4]. While this technique works well for individual, isolated memory access streams, such as a single stride, we observe that when interleaving streams this is no longer a functional approach. Instead, we propose to compute per-PC memory address deltas which maintain the delta pattern for each PC.

Due to the sheer number of memory addresses that are accessed and the fact that related memory accesses are usually spatially close to each other, using memory address deltas significantly decreased the state space of values to predict.

D. DNN Model

We cast the challenge of predicting future memory accesses as a sequence learning problem. To enable capturing the recent access history as well as longer trends we utilize an LSTM RNN model. Instead of utilizing a regression model we perform classification as we want to predict cache line aligned memory deltas. The input to the model is a sequence of PCs and memory address deltas and the output is a memory address delta relative to the absolute address that was used to compute the input delta. More specifically, the output is a probability distribution over the different classes, the most common deltas for the PCs being studied. The number of deltas is taken to be the minimum needed for 99.95% coverage of all deltas for these PCs, up to a maximum of 100. The delta with the highest probability is taken as the model's prediction. The prediction is considered correct if the next access is identical to the one that is predicted. Our model includes a single LSTM layer whose width is determined by a hyperparameter and a single dense layer that produces the output class predictions. As part of this work we vary a set of different Hyperparameters including the width of the LSTM layer and the lookback size of the LSTM. The model was trained in batches of 64 with the ADAM optimizer. We utilize Tensorflow [10] to describe our model and the Keras CuDNNLSTM layer to perform rapid hyperparameter tuning.

VI. Experiments

LSTM models were trained on compositions of the previously described memory access patterns and their accuracy is shown in Figure 1. Each model's LSTM layer had a width of 128 cells and used a lookback sequence size of 64. The first four patterns include compositions of 2-4 strided and periodic accesses. The periodic accesses which, for instance, model the traversal of an immutable list use a periodicity of 5-7. It can be seen that the LSTM performs well obtaining close to 100% prediction accuracy. In the first four patterns the interleaving of the different patterns is fixed as well, in particular, there exists a periodic sequence of individual patterns. In the case of the *4 periodic (random)* microbenchmark, the patterns individually follow a periodic sequence, however, patterns are alternated randomly. This models an application where there exist multiple independent streams of data accesses. We can observe that accuracy drops by 3.5% in this case.

For the last three microbenchmarks we add noise by inserting random accesses at random times into the regular access streams. While the address of perfectly random accesses cannot be predicted by an ML system, this experiment provides insight on how random noise affects the ability of the LSTM to predict the regular access patterns. As can be seen from

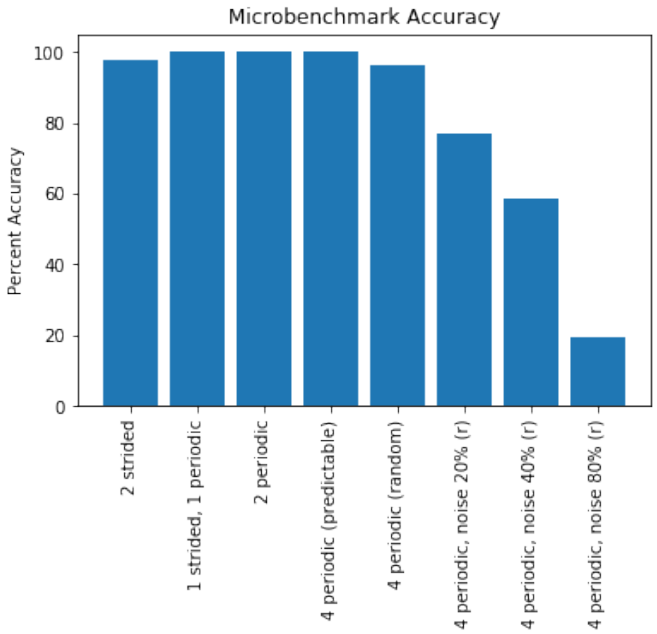

Fig. 1. Accuracy of LSTM model with microbenchmarks. Each microbenchmark is an interleaving of multiple patterns. The 4 interleaved periodic have one which has a regular predictable switch between patterns (e.g. pattern1, pattern2, pattern3, pattern4, pattern1, ...), and the rest have their next pattern chosen randomly (r). Of those a certain percentage of labelled noise is added.

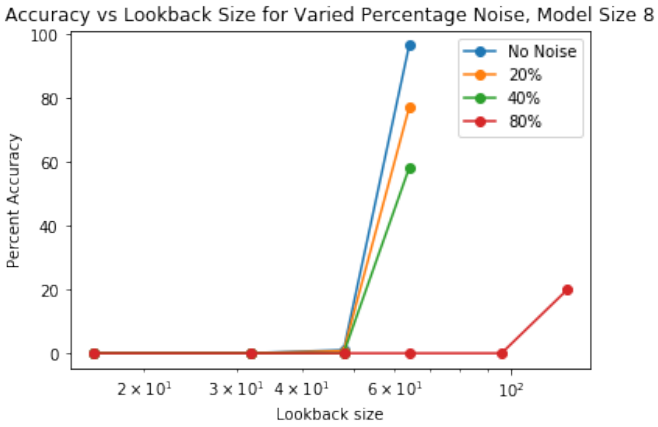

Fig. 2. Lookback size was found to have a dramatic nonlinear impact on the ability of the model to learn a pattern within the data. There appears to be some threshold lookback size below which the model is just guessing. Above this size, the model rapidly picks up the pattern. The 4 periodic patterns with varied percentage noise were used, trained on a model. An LSTM layer width of 8 cells was used; the trends were identical for larger layers.

Figure 1, adding noise created a drop in accuracy. To separate noise from signal, we added a separate 'Noise' or 'no predict' class to the model. This enables the model to distinguish noise from predictable accesses. To prevent the model training always predicting 'noise' for noiser traces, the importance of this 'no predict' class was lowered by decreasing its effect on the loss function using Keras class_weight. Since the 'noise' accesses happen randomly, the DNN understandably is unable to predict them and greater noise leads to lower accuracy. We also evaluated prediction accuracy for the regular (periodic) accesses only. In particular, we computed accuracy as the fraction of correctly predicted labels of all regular accesses in contrast to all regular and noisy accesses. The accuracy was found to be over 96% for all amounts of noise (96.3%, 97.6%, 98.2% for 20%, 40%, 80% noise respectively). These results show that the presence of labelled noise does not affect learning of separate patterns within the data access sequence.

In the following experiments, we analyze how LSTM model complexity affects the prediction accuracy. In particular, we are interested to understand the correlation between lookback size and its ability to learn streams that exhibit a long period, due to other unpredictable loads. In Figure 2 lookback size was varied to determine the impact of its choice on a model's accuracy. It was found to play an important role, where if the lookback size was too low the model was unable to learn the pattern. As soon as it reached some threshold size the model was able to achieve full expected accuracy. The threshold for the 0%, 20%, and 40% traces was between 48 and 64. For the 80% trace the threshold was between 92 and 128 accesses. The pattern was switched every 20 accesses, so 48 accesses for the no noise case is more than enough to include the full accesses from 2 patterns. It appears that the lookback size has to be much larger than the periodicity of a pattern, and larger than the number of accesses before switching to another pattern.

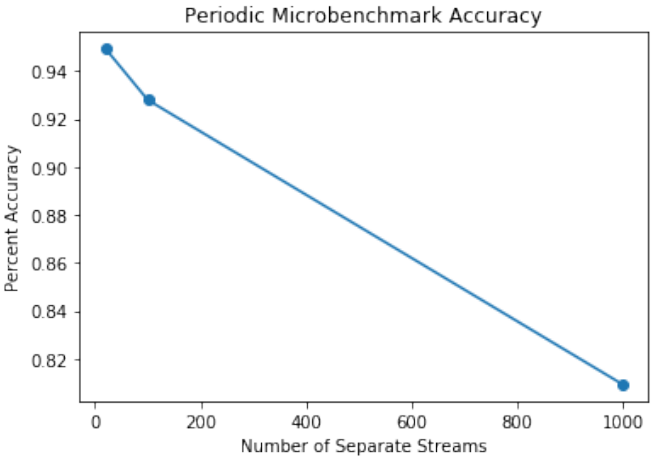

Fig. 3. Accuracy decreases as the number of interleaved streams (periodic patterns) increases.

For the microbenchmarks explored so far, a small model with LSTM layer width of 8 was sufficient to capture the pattern. To explore limits of this smaller model, the number of unique streams (interleaved patterns) was increased.

As the number of interleaved streams increases, it becomes increasingly difficult to rapidly identify the stream a particular memory access belongs to, causing an expected drop in accuracy as seen in Figure 3. The threshold for effective lookback size is unaffected by the number of streams as can be seen by the identical trend in both Figs 2 and 5. However, our experiments suggest that lookback size is related to the

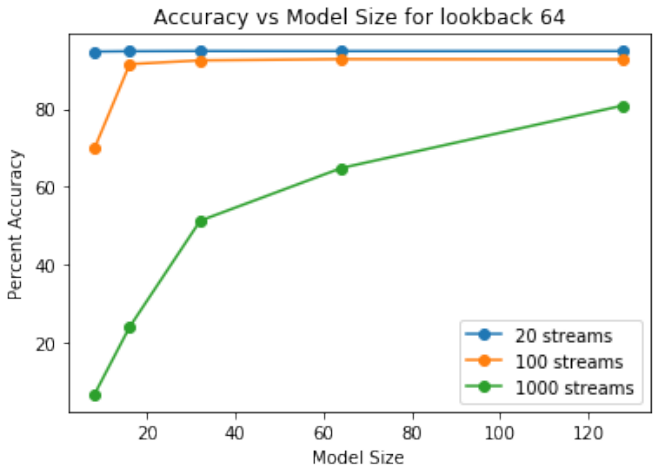

Fig. 4. Maximum accuracy for interleaved periodic streams increases as the width of the LSTM layer is increased. The importance of larger model becomes much more important as the number of distinct streams to learn is increased.

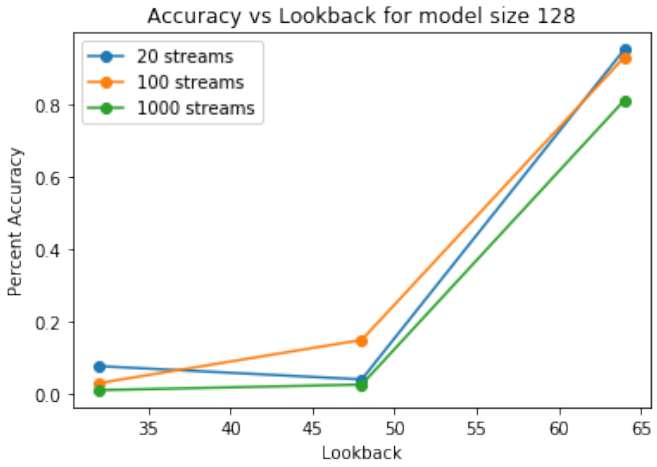

Fig. 5. For the multiple periodic streams, accuracy dramatically increases once the window size (lookback) is increased past a threshold. The same relationship holds regardless of the number of separate streams.

window size needed to capture the pattern. As predicted, the size of the model (number of parameters) becomes important as the information it must learn increases. In Figure 4, it can be seen that a larger model size is required to capture applications with 1000 or more streams. Figure 4 also shows that DNNs enable compression as the model size scales sublinear compared to the number of streams. For instance, a model width of 120 is sufficient to learn an application with 1000 streams where as conventional prefetchers that store per stream scale linearly in terms of storage resources.

For the last microbenchmark, linked list lookups were performed on lists with varied sizes providing a maximum accuracy of 99%. Interestingly, increasing lookback from 32 to 64 only increased the accuracy by about 10 percentage points as seen in Figure 6. The same dramatic "threshold" observed for the periodic microbenchmarks is not seen here. The number of deltas required to achieve near-complete coverage was much smaller than the number of nodes, probably since nodes were allocated adjacently on cache lines. This made the access pattern simpler and potentially easier to learn with a smaller window of recent memory accesses. This view is supported by the finding that increasing the model size did not provide any benefit.

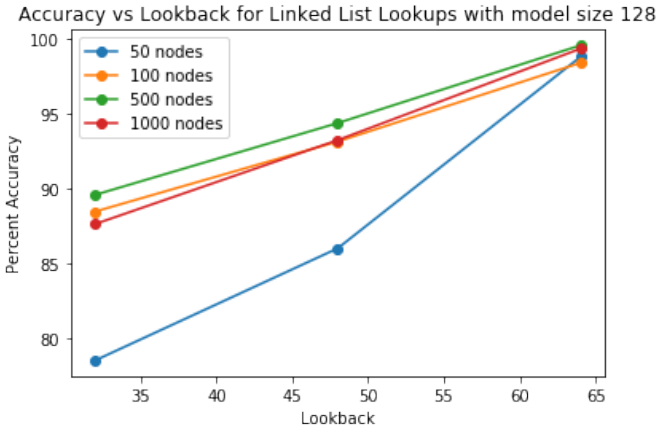

Fig. 6. Higher lookback captures the local pattern and is able to provide higher accuracy. There is not a significant difference between linked list sizes since a small number of deltas provides almost complete coverage.

VII. Related Work

Here we highlight several threads of interaction of machine learning with computer architecture.

Prior work has used a perceptron to predict whether a branch is taken or not taken [11]. The perceptron learns in an online fashion by incrementing or decrementing weights analogous to the commonly used two-bit counters.

A naive Bayesian model has been used to predict microarchitectural power and performance for more efficient design space exploration [12].

Other research also looks at learning methods to improve system performance. One system is designed to be able to manage itself, noticing changes in its environment and working to achieve global system goals such as low network latency, higher reliability, power efficiency and adaptability [14]. Another learning algorithm addresses the existence problem in a multiple-WBAN environment using a naive Bayesian classifier [13].

The explosion of interest in DNNs and machine learning has spurred parallel efforts in accelerating these models. The specialized hardware research may be typified by the TPU [16], a specialized hardware accelerator for neural network training and inference. Another interesting direction looks at eliminating spurious computations during NN prediction [13], using this in conjunction with specialized hardware to improve speed and efficiency.

VIII. Conclusion and Future Work

The recent increase in deep learning research exposes tools that have promise for being applied to microarchitectural

problems such as the pattern prediction problem of data prefetching. These applications have been minimally explored and information on how best to apply these techniques to the data prefetching problem is lacking.

We identified a starting point for evaluation of data prefetchers on simpler memory access traces that can represent components of more complex workloads. Through exploration of the model parameters, we found that lookback size must be chosen carefully to be able to capture local patterns. We found that for fewer interleaved patterns increasing model size provided no benefit, but that for many unique streams providing a larger model played a large role. We demonstrated the ability of the LSTM model to learn compositions of strided and periodic patterns, with and without added noise, and to be able to learn the patterns in a linked list traversal.

The impact of lookback window size on prediction accuracy suggests that the local patterns can be identified with just a sufficient history of memory accesses and that the long short-term memory of an LSTM may play a smaller role. Future work may explore other DNN architectures such as CNNs. Two features were used in these experiments: program counter and distance between successive memory accesses. Future work may explore the addition of more features such as previously loaded data to make it possible to maintain performance on a rapidly changing linked list as well as instructions executed preceding a memory load.

REFERENCES

[1] Ayers, Grant, Ahn, Jung Ho, Kozyrakis, Christos, and Ranganathan, Parthasarathy. Memory hierarchy for web search. In IEEE International Symposium on High-Performance Computer Architecture, HPCA-IS 18, 2018.

[2] LeCun, Yann. Bengio, Y. Hinton, Geoffrey. (2015). Deep Learning. Nature. 521. 436-44. 10.1038/nature14539.

[3] Hashemi, M., Swersky, K., Smith, J., Ayers, G., Litz, H., Chang, J., Kozyrakis, C., Ranganathan, P.. (2018). Learning Memory Access Patterns. Proceedings of the 35th International Conference on Machine Learning, in PMLR 80:1919-1928

[4] Peled, Leeor, Shie Mannor, Uri Weiser, and Yoav Etsion. "Semantic locality and context-based prefetching using reinforcement learning." In 2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA), pp. 285-297. IEEE, 2015.

[5] John L. Henning. 2006. SPEC CPU2006 benchmark descriptions. SIGARCH Comput. Archit. News 34, 4 (September 2006), 1-17. DOI=http://dx.doi.org/10.1145/1186736.1186737

[6] D. L. Bruening. Efficient, Transparent, and Comprehensive Runtime Code Manipulation. PhD thesis, M.I.T. (http://www.cag.lcs.mit.edu/dynamorio/), September 2004.

[7] Daniel Sanchez and Christos Kozyrakis. 2013. ZSim: fast and accurate microarchitectural simulation of thousand-core systems. In Proceedings of the 40th Annual International Symposium on Computer Architecture (ISCA '13). ACM, New York, NY, USA, 475-486. DOI: http://dx.doi.org/10.1145/2485922.2485963

[8] Jiwei Li, Xinlei Chen, Eduard Hovy, and Dan Jurafsky. 2016. Visualizing and understanding neural models in nlp. In Proceedings of NAACL.

[9] K. J. Nesbit and J. E. Smith, "Data cache prefetching using a global history buffer," in IEEE Micro, vol. 25, no. 1, pp. 90-97, Jan.-Feb. 2005. doi: 10.1109/MM.2005.6

[10] M. Abadi, A. Agarwal et al., Tensorflow: Large-scale machine learning on heterogeneous distributed systems, arXiv:1603.04467, 2016.

[11] D. A. Jimenez and C. Lin, "Dynamic branch prediction with perceptrons," Proceedings HPCA Seventh International Symposium on High-Performance Computer Architecture, Monterrey, Nuevo Leon, Mexico, 2001, pp. 197-206. doi: 10.1109/HPCA.2001.903263

[12] Benjamin C. Lee and David M. Brooks. 2006. Accurate and efficient regression modeling for microarchitectural performance and power prediction. SIGARCH Comput. Archit. News 34, 5 (October 2006), 185-194. DOI: https://doi.org/10.1145/1168919.1168881

[13] J. Cho, Z. Jin, Y. Han, and B. Lee, A Prediction Algorithm for Coexistence Problem in Multiple-WBAN Environment, International Journal of Distributed Sensor Networks, Vol. 2015, Article ID

[14] W. Wu and A. Louri, "A Methodology for Cognitive NoC Design," in IEEE Computer Architecture Letters, vol. 15, no. 1, pp. 1-4, 1 Jan.-June 2016. doi: 10.1109/LCA.2015.2447535

[15] M. Song, J. Zhao, Y. Hu, J. Zhang and T. Li, "Prediction Based Execution on Deep Neural Networks," 2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA), Los Angeles, CA, 2018, pp. 752-763.0 doi: 10.1109/ISCA.2018.00068

[16] N. P. Jouppi et al., "In-datacenter performance analysis of a tensor processing unit," 2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA), Toronto, ON, 2017, pp. 1-12. doi: 10.1145/3079856.3080246